APLICACIÓN WEB

GENERADOR DE CURRÍCULOS EN DIFERENTES ESTILOS

Sistemas informáticos 2009-2010

Autores:

Joaquín Castilla Carramiñana

Laura Mendiola Martínez

Rosa Olivia Zumaeta Sánchez

Profesor director:

Dr. Samir Genaim

AGRADECIMIENTOS

Como somos tres, y la lista de agradecimientos y dedicatorias podría ocupar más de lo deseado nos limitaremos a dar las gracias en primer lugar a Samir Genaim, por ofrecernos y dirigirnos un proyecto que comenzó tarde y con muchas dificultades. Gracias Samir.

Queremos agradecer a todas aquellas personas que por desgracia ya no están, pero que se sentirían muy orgullosos de nosotros.

También agradecer por supuesto a nuestros familiares y amigos por todo el apoyo brindado durante estos duros años de universidad.

Y por último agradecer a todos los que nos han apoyado con sus palabras y alegrado con nuestros progresos.

Gracias a todos.

INDICE

1. RESUMEN

Este proyecto recoge la creación de una aplicación web destinada a realizar el Currículum Vitae en diferentes formatos de forma personalizada.

Dicha aplicación servirá de soporte a distintos usuarios, que podrán gestionar sus datos profesionales de forma automatizada. Se les permitirá generar distintos formatos y estilos del mismo sin necesidad de rellenar sus datos de forma repetitiva y ajustándose a la normativa necesaria para el mismo.

Proporciona una herramienta colaborativa en la cual los diferentes usuarios podrán crear y compartir estilos y secciones del documento, manteniendo siempre la privacidad en sus datos.

Para esta aplicación se ha incluido a modo de ejemplo el formato Europeo requerido en los currículos del personal docente de la universidad UCM (Universidad Complutense de Madrid).

SUMMARY

This project includes the creation of a web application designed to make your CV in different formats in a personalized way.

This application will support different users, they can automatically manage their business data. They will generate different formats and styles of the same without filling your data on repetitive times and according to the regulations necessary for it.

It provides a collaborative tool in which different users can create and share styles and sections of the document, while maintaining privacy in their data.

For this application has been included as an example the European format required in the curricula of the faculty of the UCM University (Universidad Complutense de Madrid).

2. PALABRAS DESTACADAS

Servidor web: Es un programa que está diseñado para transferir hipertextos, páginas web o páginas HTML, formularios, botones, animaciones o reproductores de música. El programa implementa el protocolo HTTP.

Servlet: Es un componente software escrito en java que dinámicamente extiende la funcionalidad de un servidor genérico.

Máquina virtual: es una capa intermedia entre el hardware de la computadora y un software que se encarga de funciones como traducir instrucciones o simular el funcionamiento de una computadora específica.

Tomcat: Servidor de páginas JSP utilizado para procesar las páginas de la aplicación.

XSL: Es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio.

XML: Es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de una gran variedad de datos.

Látex: Sistema de preparación de documentos.

Javascript: Es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

MySQL: Gestor de base de datos que se encarga de la base de datos de una aplicación.

HTML: (*HyperText Transfer Protocol*) Lenguaje utilizado como base para la creación de páginas web.

3. DESCRIPCIÓN DE LA APLICACIÓN

3.1 OBJETIVO

El objetivo de la aplicación es la generación de Currículos en distintos formatos y estilos sin la necesidad de reescribir los datos varias veces.

Además todo debe poder hacerse en un entorno colaborativo en el cual las categorías propias de los currículos (como datos personales, formación académica....) creadas por una persona puedan ser usadas por cualquier usuario de la aplicación.

3.2 ESTRUCTURA DE LA APLICACIÓN

Podemos dividir la aplicación en varias partes bien diferenciadas.

La primera podría ser la base de datos. La web hace uso de una base de datos de MySQL para la gestión de los usuarios. Se encargará de registrar los nuevos usuarios, validar el acceso a la aplicación del mismo y otorgar los privilegios adecuados según el tipo de usuario.

La siguiente serían los archivos XML. Todo lo relacionado con currículos se almacena en archivos XML.

Hay un archivo que se encarga de la gestión de las “categorías”. En la aplicación una categoría es una entidad que costa de uno o más campos estructurados. Un ejemplo podría ser “dirección”. Podríamos definir una categoría “dirección” que estuviera formada por un campo “calle”, otro “número”, otro “piso”, etc. A su vez podría ser incluida posteriormente como parte de una categoría mayor “datos personales”. Esta categoría podría constar de un campo “nombre”, otro “apellidos”, ambos de tipo texto, y un campo “domicilio” de tipo dirección, etc.

```
<datos_personales>
    <Seccion1 nombre="nombre" tipo="texto" />
```

<Seccion2 nombre="apellidos" tipo="texto" />

<Seccion3 nombre="domicilio" tipo="dirección" />

</datos_personales>

Estas categorías son comunes a todos los usuarios y pueden ser utilizadas para definir sus currículos. Sin embargo solo los usuarios con alto nivel de privilegio pueden crearlas.

Con la creación de una categoría se crea una entrada en otro archivo XML, el encargado de gestionar los estilos de cada categoría. Los estilos son la parte fundamental de la aplicación, son los que hacen posible que el currículo se pueda generar con diferentes aspectos y en distintos formatos. Cada categoría puede tener múltiples estilos.

Los estilos son archivos XSL, fragmentos de código Latex con instrucciones XSL. El Latex es el encargado de dar el aspecto a la categoría y el XSL el responsable de rellenar los datos.

La subida de estos estilos también es responsabilidad de los usuarios con alto nivel de privilegio. Al subir un archivo se actualiza en el XML la categoría correspondiente.

<datos_personales>

<estilo>europeo</estilo>

<estilo>colorido</estilo>

</datos_personales>

Siguiendo con los XML, los currículos de cada usuario también serán XML con las categorías seleccionadas, así como los datos irán en otro archivo. Esto hace posible que solo sea necesario rellenar esos datos una vez.

Por último existirá otro archivo XML que guarda las vistas de cada usuario. Las vistas son, por decirlo de alguna manera, las posibles generaciones de currículos, por ejemplo:

<Vistas>

```
  <Presentacion>
    <datos_personales>europeo</datos_personales>
    <formacion_academica>europeo</formacion_academica>
  </Presentacion>
  <Color>
    <datos_personales>colorido</datos_personales>
    <formacion_academica>colorido</formacion_academica>
  </Color>
</Vistas>
```

Este archivo contiene dos vistas para dos generaciones distintas del currículum. Cada vista puede contener categorías y estilos diferentes.

La decisión de utilizar archivos XML, en lugar de otras estructuras como podrían ser tablas de bases de datos, vino motivada por la mayor flexibilidad a la hora de definir las categorías y los currículos sin restricciones de tamaño, por la posibilidad de anidamiento, además de por su integración y su manejo desde Java.

La parte fundamental de la aplicación son las páginas JSP, son las encargadas de ofrecer una interfaz amigable y de gestionar los archivos XML y la base de datos.

Como se explicará más adelante, existen páginas para registrar nuevos usuarios, acceder al sitio, gestionar las categorías, los currículos, las vistas y procesar la generación del currículum propiamente dicho exportándolo por ejemplo en PDF.

Por último la aplicación hace uso de compiladores de Latex para generar los archivos definitivos a partir de los estilos.

4. PÁGINAS JSP

Entre la lista de tecnologías disponibles para la creación de páginas dinámicas de servidor se encuentran las páginas JSP (Java Server Pages). Su mayor diferencia con el resto de tecnologías existentes es que utilizan el lenguaje de programación Java, y es que las páginas JSP permiten al programador incluir código Java al igual que si se tratara de una aplicación corriente dentro de una página web para, posteriormente, ser vistas a través de un navegador.

4.1 JAVA

A estas alturas nadie puede dudar de la potencia y versatilidad del lenguaje de programación Java. Dado a conocer por Sun Microsystems haya por el año 1995.

Una de las características más importantes de este lenguaje de programación es su independencia de la plataforma en la que se vaya a ejecutar. Es decir, que cualquier aplicación programada en Java es capaz de funcionar al 100% en cualquier otra plataforma con cualquier otro sistema operativo diferente en el que fue creada.

Esto permite al programador, por ejemplo, crear su programa en una ordenador con un SO Windows, para luego ser ejecutado en una estación Sun con un sistema operativo Solaris. Esto es posible gracias a la disponibilidad de la máquina virtual de Java (JVM) para la mayoría de los sistemas operativos (Windows, Apple, Solaris, etc.) con respecto al lenguaje propiamente dicho, los conocedores de otros lenguajes como C++ encontrarán grandes similitudes con Java.

Java, en un principio tan sólo fue pensado como lenguaje de programación independiente, para crear aplicaciones que se ejecutaban en estaciones de trabajo o consolas ya fuera en modo de texto o en modo gráfico utilizando las librerías Swing y AWT. Dada la necesidad de llevar este lenguaje hasta las puertas de Internet y, por tanto, utilizarlo para crear aplicaciones que pudieran funcionar en este marco se crearon los servlets.

Un servlet es un programa que corre en un servidor y que devuelve su resultado al cliente en forma de código HTML. Muchos lo definen como un applet pero

que se ejecuta en el servidor en vez de hacerlo en el cliente. Este tipo de programas que se ejecutan en el servidor también son capaces de recoger una serie de datos como puede ser a través de un formulario y procesarlos, además de utilizar conexiones de bases de datos, mantener los estados de sesión de cada usuario, etc.

Para poder trabajar con servlets es necesario utilizar un servidor web capaz de procesar este tipo de programas para una vez compilado, devolver el código HTML al navegador. Su funcionamiento es muy similar al de las páginas ASP de Microsoft, por ejemplo, en el que una vez interpretado y ejecutado todo su código (VBScript y/o JScript) se envía junto a las etiquetas HTML al usuario.

Aquí encontramos la primera gran ventaja de las especificaciones servlet: el intérprete de este tipo de aplicaciones es capaz de diferenciar cuando el fichero ha cambiado y cuando no. Es decir, si el servlet es solicitado por un cliente, será interpretado y su resultado será enviado al navegador. A partir de ese momento el servlet permanece en memoria, si un segundo usuario solicita ese mismo servlet el intérprete verá que ya está compilado por lo que tendrá la mitad del trabajo hecho y el tiempo en enviar el resultado será mucho menor. Si el servlet se modifica, el servidor comparará la versión que tiene en memoria con la nueva, volviendo a compilar la última versión al ser la más reciente. Como el lector puede suponer esta gran ventaja redunda en el tiempo de proceso del envío de la página final, ya que en la mayoría de los casos los servlets estarán en memoria.

En la siguiente figura se puede ver cómo es la estructura básica de un servlet. Lo primero es importar los paquetes que son necesarios para crear un servlet: java.io.*, para la clase PrinterWriter; javax.servlet.*, para HttpServlet y java.servlet.http, para los interface HttpServletResponse y HttpServletRequest.

Estructura básica de un servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request,HttpServletResponse

response)
```

```
throws ServletException, IOException {
PrintWriter out = response.getWriter();
out.println("Esto es un servlet de prueba");
}}
```

Para que una aplicación Java sea un servlet debe de heredar de la clase HttpServlet y sobrescribir los métodos doPost y doGet dependiendo si los datos son enviados utilizando el método GET o POST. En cualquier caso estos métodos toman dos argumentos: HttpServletRequest y HttpServletResponse. Mediante los métodos disponibles por HttpServletRequest es posible conocer información que entra al servlet como los campos enviados desde un formulario, y a través de los métodos de HttpServletResponse es posible enviar información de respuesta al cliente, siendo el caso más utilizado escribir contenido o algún tipo de resultado al código HTML que finalmente será leído por el navegador. En este caso se utiliza la clase PrinterWriter que permite enviar la información al cliente. Cómo se puede observar en el listado los métodos doPost y doGet lanzan excepciones (ServletException y IOException) que deben de ser recogidas en el método.

En la siguiente figura se puede observar como se ha construido una página HTML utilizando un servlet. En este caso el ejemplo es sencillo ya que todos los tags se han guardado en la variable de tipo String outHTML que posteriormente será enviada al código mediante el método println() del objeto out antes creado.

Página HTML utilizando un servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
PrintWriter out = response.getWriter();
String outHTML = "";
outHTML += "<html>";
```

```
outHTML += "<head><title>Servlet</title></head>";
outHTML += "<body>";
outHTML += "<b>Esto es una página HTML creada mediante un
servlet.</b>";
outHTML += "</body>";
outHTML += "</html>";
out.println(outHTML);
}}
```

Un ejemplo de cómo una página HTML creada mediante un servlet se puede beneficiar de las ventajas de la programación Java es la siguiente figura. En él se puede comprobar cómo se ha utilizado la clase Date (dentro de paquete java.util.*) para obtener la fecha del momento en el que se solicita el servlet, de forma que una vez procesado el cliente sólo ve el código HTML resultante.

Usando Java en una página HTML

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
PrintWriter out = response.getWriter();
String outHTML = "";
Date fecha = new Date();
outHTML += "<html>";
outHTML += "<head><title>Servlet</title></head>";
outHTML += "<body>";
outHTML += "<b>Esto es una página HTML creada mediante un
servlet.</b>";
```

```
outHTML += "<br>Hoy es... " + fecha;
outHTML += "</body>";
outHTML += "</html>";
out.println(outHTML);
}}
```

4.2 LOS SERVLETS Y JSP

En el apartado anterior se ha hecho un rápido repaso a cómo funciona un servlet y cómo se comporta cuando es invocado a través de un navegador y de esta forma construir páginas dinámicas en el servidor. Pero, si bien es un sistema cómodo, tiene algunos inconvenientes.
La desventaja más importante es que no es posible separar la parte de lógica de la de presentación, es decir, si se desea cambiar un tipo de letra o añadir una columna más a una tabla ya creada es necesario modificar el fichero fuente para hacer estos cambios, luego volverlo a compilar y comprobar de nuevo el resultado. Si el cambio no ha sido correcto será necesario realizar de nuevo todos los pasos anteriores hasta lograr la modificación que se desea. Por otro lado, si los cambios afectan a la lógica de funcionamiento (accesos a bases de datos, bucles, condiciones, etc.) hay que prestar especial atención a que la página mantenga su formato y aspecto.

Es decir, si llevamos esta situación al mundo real estaríamos hablando de que un programador Java deberá de tener amplios conocimientos sobre el diseño de páginas web o que un diseñador de páginas web debe saber programación Java. Realmente, esta situación no se da con demasiada frecuencia por lo que es necesario recurrir a otra solución: las páginas JSP.

Una página JSP está basada en la especificación de los servlet. Para dar una idea aproximada se puede decir que una página JSP es un servlet con un comportamiento algo especial. Si se observa el código que forma una página JSP se observa que es prácticamente igual a una página HTML excepto por las etiquetas <% y %> (al igual que en la programación de páginas ASP de Microsoft) que dan paso a unas instrucciones escritas en Java. En la siguiente figura se muestra el ejemplo utilizado en la figura anterior, pero utilizando esta vez una página JSP. En este caso un posible cambio ya sea de programación como de presentación

podría llevarse a cabo sin ninguna dificultad porque a fin de cuentas se trata de una página web con código Java insertado.

JSP en página HTML

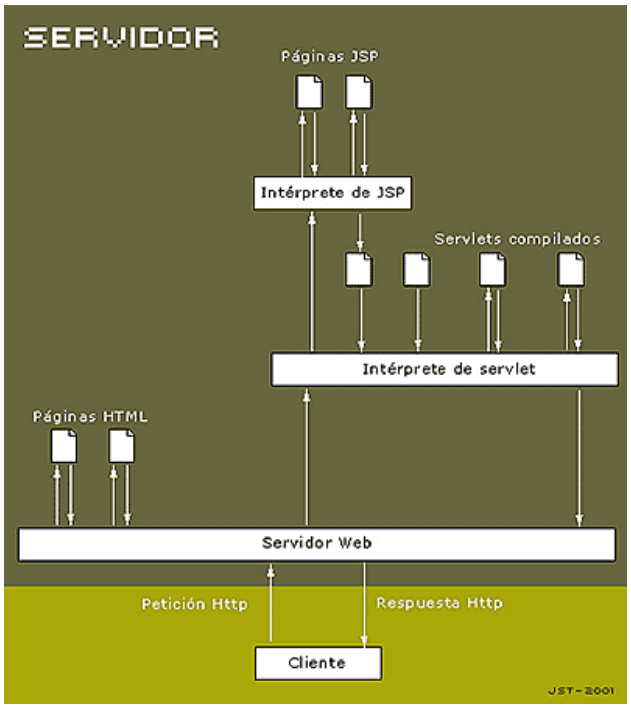
```
<%@ page import="java.util.*"%>
<html>
<head>
<title> Página JSP</title>
</head>

<body>
<b>Esto es una página HTML creada mediante un servlet.</b>
<br>
<%
Date fecha = new Date();
out.println("Hoy es... " + fecha);
%>
</body>
</html>
```

Una página cuando es solicitada por un navegador es interpretada y posteriormente se convierte en un servlet con todos sus métodos y paquetes importados necesarios para su correcto funcionamiento. Pero de esta conversión no tiene que preocuparse el programador ya que es función del servidor/intérprete de páginas JSP que se esté utilizando. Obviamente el proceso por el cual una página JSP se convierte en una página HTML necesita de algo más de tiempo al tener que pasar por dos intérpretes diferentes (el propio de JSP y posteriormente el de servlets), mientras que un servlet sólo se interpreta una vez. Al igual que se ha dicho anteriormente para los servlets, las páginas JSP tienen en mismo comportamiento a la hora de estar almacenadas en memoria y así poder ser servidas de forma más rápida.

4.3 SERVIDORES DE SERVLETS Y JSP

Para poder interpretar una página JSP cuando es solicitada por un cliente es necesario instalar y configurar un servidor adecuado para esta tecnología. En la actualidad este tipo de servidores se pueden dividir en dos grupos: interpretes de páginas JSP y Servlets y por otro lado servidores de aplicaciones. En ambos casos incluyen un servidor http capaz de atender las peticiones que le llegan a través del puerto 80 (u otro que se defina), por lo que no es necesario instalar un segundo servidor.

El primer tipo de servidores sólo desempeñan el papel de interpretar las páginas JSP o los servlets que le son solicitados. Dos de los servidores de este tipo más conocidos son Apache-Tomcat (este que hemos utilizado para el proyecto), GlassFish y Web Server de Sun. En ambos casos y después de realizar su instalación basta con alojar la aplicación JSP en un directorio y realizar una pequeña configuración para que las páginas JSP ya estén disponibles.



Un servidor de páginas JSP está compuesto por un intérprete de páginas JSP, uno de servlets y un servidor HTTP para recoger la petición de páginas.

El segundo tipo corresponden a los servidores de aplicaciones (Application Servers), que tienen como objetivo crear un entorno de ejecución para aplicaciones

de gran envergadura, además de proporcionar una serie de servicios añadidos (pool de conexiones a una base de datos, etc.) para facilitar la programación de la aplicación. Estos servidores de aplicaciones están orientados por completo hacia un entorno Java utilizando como elementos clave los EJB (Enterprise Java Beans) además de Java Beans y otros componentes. En todos los casos la capa de presentación y interacción con el software de servidor se realiza también utilizando JSP, por lo que también dispone de un intérprete para este tipo de páginas dinámicas. Algunos de los servidores de aplicaciones más conocidos (y costosos) son Weblogic de BEA y WebSphere de IBM. Aunque también hay otros como Jrun de Allaire o Orion Server. En cualquier caso, debe existir una decisión bien argumentada y sostenible para utilizar un servidor de aplicaciones ya que el desarrollo de una aplicación bajo este tipo de entorno no es nada fácil además de requerir unos amplios conocimientos en este tipo de servidores y en EJB's.

Para la puesta en marcha de una aplicación web basada en la tecnología JSP bastaría con utilizar un servidor como Tomcat. Este servidor está creado por el mismo equipo que desarrolla el conocido servidor web Apache, y puede descargarse de forma gratuita desde su página web http://tomcat.apache.org/. También está disponible para los principales sistemas operativos (Windows, Solaris, Unix, etc.) además de ser muy fácil su instalación y configuración.

Hay que añadir que para el correcto funcionamiento de un servidor de páginas JSP es necesario haber instalado previamente el entorno de desarrollo de Java JDK (Java Developer Kit) y que éste se encuentre correctamente configurado.

4.4 LAS SESIONES

El uso de sesiones es posible mantener una serie de estados y de información asociada a cada usuario que visita una página JSP. De esta forma podrá disponer de él en cualquier página sin que su valor (contenido) se pierda en ningún momento.

Para utilizar las sesiones en las páginas JSP se utiliza el interface HttpSession incluido dentro de la API de servlets. Esta interfaz dispone de una serie de métodos para poder utilizar las sesiones. Para poder utilizar una sesión en una página JSP el primer paso es crear una instancia de HttpSesion.

HttpSession miSesion = request.getSession();

Una vez creada la instancia se puede obtener el identificador de la sesión que ha sido asignado por el servidor al usuario la primera vez que ha visitado alguna de las páginas JSP. Este ID es único para cada usuario de forma que se crearán tantos usuarios como haya conectados a la aplicación. Si un usuario ya tiene asignado un identificador de sesión, éste se mantendrá durante toda la conexión. Para poder recuperar este identificador, se utiliza el método getId().

String idMiSesion = miSesion.getId();
out.println ("El identificador de la sesión es: " + idMiSesion);

Dependiendo del tipo de servidor empleado, este identificador será diferente, por ejemplo un ID de sesión generado por Tomcat será distinto al generado por Weblogic. De entre todos los métodos disponibles para conocer información acerca de la sesión cabe destacar algunos como getCreationTime(), para saber cuándo fue creada la sesión; getLastAccessedTime() para conocer cuándo fue la última vez que se envió una respuesta al cliente asociada a la sesión, etc.

En la siguiente figura se muestra un ejemplo de cómo utilizando estos dos métodos se puede conocer cuánto tiempo ha permanecido un usuario visitando el website.

Ejemplo de uso de sesión

```
<%@ page import="java.util.*"%>
<html>
<head>
<title>Página JSP</title>
</head>

<body>
<%
HttpSession miSesion = request.getSession();
String idMiSesion = miSesion.getId();
long sesionCreada = miSesion.getCreationTime();
long sesionUltima = miSesion.getLastAccessedTime();
```

```
long sesionDuracion = sesionUltima - sesionCreada;
out.println ("El id de la sesión es: " + idMiSesion + "<br>");
out.println ("Sesión creada: " + new Date(sesionCreada) + "<br>");
out.println ("Última sesión: " + new Date(sesionUltima) + "<br>");
out.println ("Duración de la sesión: " + (new
Date(sesionDuracion)).getMinutes());
%>
</body>
</html>
```

Trabajar con sesiones en una página JSP permite, como se ha dicho anteriormente, asociar un objeto a una sesión, es decir a un usuario, de forma que ese objeto sólo le pertenecerá a él mientras dure esa sesión.

```
HttpSession miSesion = request.getSession();
Objeto miObjeto = new Objeto();
miObjeto.addProducto(disco);
miObjeto.addProducto(libro);
miObjeto.addProducto(dvd);
miSesion.setAttribute("objeto",miObjeto);
```

4.5 DIRECTIVAS DE JSP

Para la creación de páginas JSP, además del propio lenguaje Java, es necesario conocer y utilizar algunas etiquetas que nos permiten "dar órdenes" al intérprete de páginas JSP además de comunicarle una serie de acciones a realizar.

Estas acciones se diferencian entre directivas y acciones estándar. Las directivas son utilizadas para establecer valores globales como la declaración de una clase, métodos a implementar, etc. Estos mensajes son enviados al contenedor de JSP desde la propia página de JSP. Estas directivas no dan como resultado ningún mensaje al cliente y su alcance es únicamente a la página JSP. Las directivas se caracterizan por contener el símbolo "@" en su comienzo y suelen tener la siguiente sintaxis:

<%@ nombre_directiva atributo1="valor1" atributo2="valor2"... %>

En JSP existen tres directivas: page, include y taglib.

Page define el número de propiedades y atributos que van a afectar directamente a la página JSP. Sus sintaxis, con algunos de los parámetros más utilizados es la siguiente:

<%@ page language | extends | import | session | isErrorPage | errorPage = "valor" %>

Cada uno de estos atributos define un valor que afecta directamente a la ejecución de la página JSP. El atributo language establece el lenguaje que se utiliza, en este caso se trata de "Java". Extends, establece que la clase generada a partir de dicha página JSP debe de "heredar" de una clase superior. Import define los paquetes o clase que deben ser importados. Session especifica si la página utiliza sesiones HTTP. El atributo isErrorPage establece si se va a utilizar la característica de mostrar una página de error definida por el programador, pudiendo mostrar información acerca del mismo o qué lo produjo, el valor que podrá tomar, por lo tanto es "true" o "false". Para que esta característica pueda funcionar correctamente es necesario que el atributo errorPage contenga la dirección URI y nombre de la página a mostrar.

El siguiente ejemplo muestra la directiva page con los atributos antes visto y sus posibles valores:

<%@ page language="java" extends="superclase" import="java.util." session="true" isErrorPage="true" errorPage="pagina_error.jsp" %>*

La segunda directiva que JSP proporciona es el conocido **include**, y avisa al contenedor que debe de incluir el contenido de un fichero en la página JSP que contiene esta directiva. Lógicamente dicha página debe de estar disponible en el servidor. La sintaxis del uso de esta directiva es la siguiente:

<%@ include file="nombre_fichero" %>

El contenido del fichero incluido mediante esta directiva será interpretado por la página JSP y esto sólo ocurre en tiempo de compilación (como se verá más tarde hay una acción include que se ejecuta en tiempo de ejecución). El fichero incluido no debe ser otra página dinámica. Además, la mayoría de los contenedores de JSP llevan un seguimiento de cuando un fichero incluido ha sido modificado para recompilar la página JSP de nuevo.

La siguiente línea muestra un ejemplo del uso de la directiva include con el objetivo de añadir una página HTML a la página JSP que realiza la llamada:

<%@ include file="inc/otra_pagina.html" %>

La directiva es **taglib**, que permite incluir una librería de etiquetas propias creadas por el programador y que pueden ser utilizadas a lo largo de toda la página JSP. Su sintaxis es la siguiente:

<%@ taglib uri = "tagLibraryURI" prefix="tagPrefix" %>

Hasta ahora las etiquetas que se utilizan son del tipo "<jsp:parametro="valor" />", pero también es posible crear librerías de etiquetas si el programador lo cree oportuno con el fin de hacer más fácil la programación de la página.

Un ejemplo de estas etiquetas podrían ser:

<iworld: articulo="Páginas JSP" />
<iworld: autor="Esteban" />

Para utilizar estas etiquetas, la directiva taglib se utiliza de la siguiente forma:

<%@ taglib uri = "http:// www.servidor.com/misEtiquetas" prefix="iWorld" %>

En este ejemplo, el parámetro uri (Uniform Resource Indentifier) establece el lugar en el que se encuentra el descriptor de la librería de etiquetas. Este descriptor se encarga de notificar al contenedor de páginas JSP qué debe de hacer cuando encuentre una etiqueta especificada. El parámetro prefix, define el prefijo utilizado en ese conjunto de etiquetas, en este ejemplo "iWorld".

4.6 Etiquetas estándar de JSP

Al contrario que ocurre con las directivas, JSP dispone de unas etiquetas que si afectan al comportamiento de la página JSP en tiempo de ejecución y al resultado de la página que le llega al usuario. La especificación JSP establece una serie de acciones estándar y que pueden ser ampliadas por terceras empresas con el fin de lograr una mejora en sus intérpretes y contenedores de JSP.

Las etiquetas estándar son tres: jsp:include, jsp:param y jsp:forward. La primera de ellas, es similar a la directiva include vista anteriormente, con la diferencia de

que esta vez si es posible utilizar páginas estáticas o dinámicas en el momento en el que se solicita una página JSP (request time). Su sintaxis es la siguiente:

<jsp:include page="nombre_pasgina.jsp" flush="true" />

Se podrá observar en el fichero java que se genera una línea similar a la siguiente:

pageContext.include(pagina.html);

Y en el caso de utilizar la directiva include, el resultado sería:

out.print("Esto es un fichero include");

La acción jsp:param se utiliza para añadir información adicional a otras etiquetas mediante la forma atributo/valor.

<jsp:param name="nombre_parametro" value="valor_parametro" />

Esta acción es utilizada junto a las etiquetas jsp:include, jsp:forward y jsp:plugin.

En las siguientes líneas se puede ver un ejemplo de cómo utilizar de forma conjunta las acciones include y param.

```
<jsp:include page="portada.jsp">
<jsp:param name="id1" valor="noticias">
<jsp:param name="id2" valor="nacional">
</jsp:include>
```

En este ejemplo, la página portada jsp puede hacer uso de los valores incluidos (id1 e id2) mediante el método request .getParameter("id1") y request.getParameter("id2").
Por último, la acción jsp:param permite que una solicitud de página JSP sea redirigida hacia otra página distinta. Su sintaxis es:

<jsp:forward page="nombre_pagina.jsp" />

Como se ha visto en el caso de la acción include, jsp:forward también puede ser utilizada junto a la etiqueta jsp:param para añadir parámetros.

```
<jsp:forward page="nueva_portada.jsp">
<jsp:param name="id1" value="deportes">
<jsp:param name="id2" value="hockey">
</jsp:forward>
```

4.7 JavaBeans

La programación de páginas JSP permite utilizar un bloque de código que tiene encomendada una funcionalidad específica. JSP permite crear JavaBeans que podemos definir como una caja negra que de la que no es necesario conocer cómo funciona o qué tiene, ya que con sólo saber qué necesita, qué devuelve y qué hace será suficiente.
La ventaja que ofrece utilizar estos JavaBeans es que permiten independizar, ya sea a la hora de crear la página JSP como de definir las diferentes funcionalidades que va a tener la aplicación Web, las tareas de programación: mientras que un programador Java puede construir un JavaBean para recoger, actualizar y grabar información en una base de datos un programador web puede crear la página que será el interface que utilizarán los usuarios.

La similitud de los JavaBeans se encuentra en la tecnología de ASP bajo en nombre de objetos COM+.

Para utilizar un JavaBean en una página JSP hay que hacer uso de tres etiquetas de tipo estándar como las vistas anteriormente. La primera de ellas, y más importante, es la que realiza la carga de JavaBean en la página JSP su sintaxis es:

<jsp:useBean class="JavaBean"
id="nombre_instancia_JavaBean"
scope="page|request|session|application" />

En la siguiente línea se puede ver un ejemplo del uso de la etiqueta jsp:useBean:

<jsp:useBean class="pruebas.calendario" id="cal" scope="page" />

Los parámetros que requiere son el nombre del JavaBean, el nombre de la instancia a la que se hará referencia cuando sea utilizado, y por último, el ámbito que va a tener.
Para incluir datos de entrada una vez cargado en JavaBean en la página JSP se utiliza la etiqueta:

<jsp:setProperty name="nombre_instancia_JavaBean" nombrePropiedad="valor" />

La siguiente línea toma el valor de las variables día, mes, año para incluirlas en el JavaBean antes cargado:

<jsp:setProperty name="cal" property="initDay" value="<%=dia%>" />

<jsp:setProperty name="cal" property="initMonth" value="<%=mes%>" />
<jsp:setProperty name="cal" property="initYear" value="<%=anio%>" />

Por último, para recuperar un resultado producido por el JavaBean, se utiliza la etiqueta jsp:getProperty:

<jsp:getProperty name="nameBean" property="propertyName" />

En la siguiente línea se recupera el resultado de JavaBean que en este caso corresponde a todo el código necesario para dibujar el calendario.

<jsp:getProperty name="cal" property="dibujarCalendario" />

La construcción de una JavaBean se realiza de forma similar a cualquier clase de Java, con la salvedad que debe de utilizar el prefijo get para los métodos que devuelven algún tipo de valor y el prefijo set para aquello que reciben algún valor a través de la etiqueta jsp:setProperty.

La siguiente figura muestra una clase que actúa como JavaBean. En este caso utiliza un método set (guardarNombre) para almacenar el nombre que le llega por medio de la etiqueta jsp:setProperty y un método get (recuperarNombre) para devolver el nombre antes guardado. Observe el lector como en la primera línea se ha incluido la clase dentro de un paquete llamado pruebas; y que posteriormente al carga el JavaBean en la página JSP se deberá indicar este paquete.

JavaBean (Declaración)

```
package pruebas;

public class nombre {
String unNombre;

public void setGuardarNombre(String unNombre() {this.unNombre =
unNombre;
}

public String getRecuperarNombre() {
return unNombre;

}}
```

Una vez compilada y colocada en un directorio que esté accesible mediante un CLASSPATH previamente definido, se utiliza el JavaBean como se muestra a continuación.

JavaBean (Uso)

```
<jsp:useBean id="nomb" class= "pruebas.nombre" scope="page"/>
<html>
<head>
<title>Probando un JavaBean</title>
</head>
<body>
<%
String nombre = "Esteban";
%>
Vamos a guardar el nombre "<%=nombre%>" en el JavaBean.<br>
<jsp:setProperty name="nomb" value="<%=nombre%>" proper-
ty="guardarNombre"/>
Nombre que se ha guardado: <jsp:getProperty name="nomb"
property="recuperarNombre"/>
</body>
</html>
```

4.8 RECOGER DATOS DESDE UN FORMULARIO

La tecnología JSP también permite interactuar con los datos que llegan al servidor por medio de los formularios. Cuando un usuario rellena un formulario y en enviado mediante un evento submit al servidor utilizando cualquiera de los dos métodos disponibles (POST y GET) los datos son encapsulados bien en la cabecera del protocolo HTTP (en el caso del método POST) o junto a la URL (en el caso de GET). En ambos casos estos datos enviados pueden ser recogidos para posteriormente ser tratado por la aplicación. Para poder recoger estos datos se utiliza el método getParameter del objeto request, utilizando como argumento el nombre del campo que se desea recoger. El siguiente ejemplo muestra como se recoge un campo llamado nombre de un formulario que ha sido enviado al servidor:

out.println ("Tu nombre es: " + request.getParameter("nombre"));

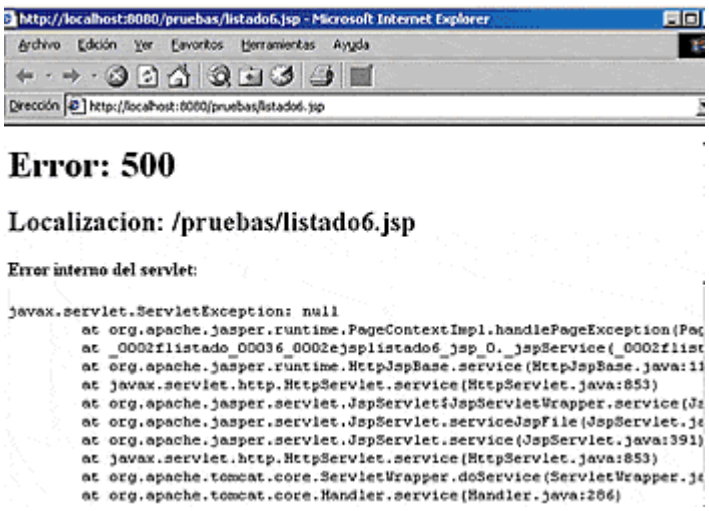
Aunque una forma más clara de realizar esta tarea es declarando una variable de tipo String que recogerá dicho valor del formulario:

String nombre = request.getParameter("nombre");
out.println ("Tu nombre es: " + nombre);

Un punto importante es que todo los datos que son recogido mediante este sistema serán de tipo String a pesar de que se trata de números, por ello si una de las operaciones es la de realizar algún tipo de cálculo, será obligatorio convertir ese número de tipo String a un valor de tipo numérico.

String nombre = request.getParameter("nombre");
String edad = request.getParameter("edad");
int edadInt = Integer.parseInt(edad);
out.println ("Tu nombre es: " + nombre + " y tienes " + edad + " años.");

Hay que prestar especial cuidado a este tipo de conversiones ya que si alguno de los valores recibidos en de tipo null, la conversión no podrá llevarse a cabo y el intérprete devolverá un mensaje de error.



Pantalla de error provocado al intentar operar con un valor null dentro de un servlet.

En la siguiente imagen se puede ver este mensaje de error al ser de tipo null el valor que se pasa al método parseInt().

Para poder mostrar un formulario y posteriormente procesar sus datos no es obligatorio utilizar dos páginas diferentes. Utilizando la sentencia de condición if es posible diferenciar cuando se ha pulsado el botón de enviar (submit) o cuando no, mostrando por pantalla el formulario vacío o el resultado.

La siguiente figura muestra como en una misma página se piden unos datos a través de un formulario para luego ser mostrados desde la misma página. En esta ocasión el valor del parámetro activo del formulario es un método del objeto request que devuelve la página en la que está dicho método. La condición para que muestre o no el formulario es el valor que tome el campo "nombre". Cómo es lógico al entrar por primera vez el valor de dicho campo será null, por lo que deberá demostrar el formulario; cuando el usuario pulse el botón enviar, este campo ya tendrá un valor, por lo que deberá ejecutarse el bloque else de la sentencia if, en la que se recoge y muestra la información enviada.

Recogida de datos en la misma página

```
<html>
<head>
<title>Página JSP</title>
</head>

<body>
<% if (request.getParameter("nombre") == null) { %>
<form action="<%=request.getRequestURI()%>" method="post">
Nombre: <input type="text" name="nombre" size="20"><br>
Edad: <input type="text" name="edad" size="4"><br>
<input type="submit" name="Enviar">
</form>
<%
} else {
String nombre = request.getParameter("nombre");
String edad = request.getParameter("edad");
int edadInt = Integer.parseInt(edad);
out.println( "Tu nombre es: " + nombre + " y tienes " + edad + "
años.");
```

```
}
%>
</body>
</html>
```

Dentro del objeto request existen varios métodos que pueden resultar de gran importancia al programador de páginas JSP. Uno de ellos es getHeader() que, utilizando como argumento la cadena "User-Agent", devuelve el tipo de navegador utilizado; o los métodos RemoteHost() y ServerName que devuelve los nombres del ordenador cliente que realiza la petición de una página JSP y del servidor, respectivamente.

WEB DE INTERÉS

- Especificación oficial de Servlet java.sun.com/products/servlet
- Especificación oficial de JSP java.sun.com/products/jsp
- JSP Resource Index www.jspin.com

5. XML

XML, siglas en inglés de *Extensible Markup Language* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

5.1 INTRODUCCIÓN

XML proviene de un lenguaje inventado por IBM en los años setenta, llamado GML (*Generalized Markup Language*), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje gustó a la ISO, por lo que en 1986 trabajaron para normalizarlo, creando SGML (*Standard Generalized Markup Language*), capaz de adaptarse a un gran abanico de problemas. A partir de él se han creado otros sistemas para almacenar información.

En el año 1989 Tim Berners Lee creó la web, y junto con ella el lenguaje HTML. Este lenguaje se definió en el marco de SGML y fue de lejos la aplicación más conocida de este estándar. Los navegadores web sin embargo siempre han puesto pocas exigencias al código HTML que interpretan y así las páginas web son caó-

ticas y no cumplen con la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos.

Otra limitación del HTML es que cada documento pertenece a un vocabulario fijo, establecido por el DTD. No se pueden combinar elementos de diferentes vocabularios. Asimismo es imposible para un intérprete (por ejemplo un navegador) analizar el documento sin tener conocimiento de su gramática (del DTD). Por ejemplo, el navegador sabe que antes de una etiqueta debe haberse cerrado cualquier <p> previamente abierto. Los navegadores resolvieron esto incluyendo lógica ad hoc para el HTML, en vez de incluir un analizador genérico. Ambas opciones, de todos modos, son muy complejas para los navegadores.

Se buscó entonces definir un subconjunto del SGML que permita:

- Mezclar elementos de diferentes lenguajes. Es decir que los lenguajes sean extensibles.
- La creación de analizadores simples, sin ninguna lógica especial para cada lenguaje.
- Empezar de cero y hacer hincapié en que no se acepte nunca un documento con errores de sintaxis.

Para hacer esto XML deja de lado muchas características de SGML que estaban pensadas para facilitar la escritura manual de documentos. XML en cambio está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

5.2 VENTAJAS DEL XML

- Es extensible: Después de diseñado y puesto en producción, es posible extender XML con la adición de nuevas etiquetas, de modo que se pueda continuar utilizando sin complicación alguna.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada versión de lenguaje XML. Esto posibilita el empleo de cualquiera de los analizadores disponibles. De esta manera se evitan *bugs* y se

acelera el desarrollo de aplicaciones.

- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarla. Mejora la compatibilidad entre aplicaciones.

5.3 ESTRUCTURA DE UN DOCUMENTO XML

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de pedazos de información. Ejemplos son un tema musical, que se compone de compases, que están formados a su vez por notas. Estas partes se llaman *elementos*, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma **<*nombre*>**, donde *nombre* es el nombre del elemento que se está señalando.

A continuación se muestra un ejemplo para entender la estructura de un documento XML:

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--se utiliza UTF-8 debido a que acepta casi todo tipo de caracteres el cual es recomendaro-->

<!DOCTYPE Edit_Mensaje SYSTEM "Lista_datos_mensaje.dtd"

                    [<!ELEMENT Edit_Mensaje (Mensaje)*>]>

<Edit_Mensaje>

    <Mensaje>

        <Remitente>

            <Nombre>Nombre del remitente</Nombre>

            <Mail> Correo del remitente </Mail>
```

```
        </Remitente>

        <Destinatario>
            <Nombre>Nombre del destinatario</Nombre>
            <Mail>Correo del destinatario</Mail>
        </Destinatario>
        <Texto>
            <Asunto>
                Este es mi documento con una estructura muy sencilla
                no contiene atributos ni entidades....
            </Asunto>
            <Parrafo>
                Este es mi documento con una estructura muy sencilla
                no contiene atributos ni entidades....
            </Parrafo>
        </Texto>
    </Mensaje>
</Edit_Mensaje>
```

5.4 DOCUMENTOS XML BIEN FORMADOS

Los documentos denominados como "bien formados" (del inglés *well formed*) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, analizarse correctamente por cualquier analizador sintáctico (*parser*) que cumpla con la norma. Se separa esto del concepto de validez que se explica más adelante.

- Los documentos han de seguir una estructura estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.

- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.

- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.

- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.

- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.

- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos "entendibles" por las personas.

5.5 PARTES DE UN DOCUMENTO XML

Un documento XML está formado por el prólogo y por el cuerpo del documento así como texto de etiquetas que contiene una gran variedad de efectos positivos o negativos en la referencia opcional a la que se refiere el documento, hay que tener mucho cuidado de esa parte de la gramática léxica para que se componga de manera uniforme.

5.6 PRÓLOGO

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.

El prólogo de un documento XML contiene:

- Una declaración XML. Es la sentencia que declara al documento como un documento XML.
- Una declaración de tipo de documento. Enlaza el documento con su DTD (definición de tipo de documento), o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- Uno o más comentarios e instrucciones de procesamiento.

5.7 CUERPO

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener un y solo un elemento raíz, característica indispensable también para que el documento esté bien formado. Sin embargo es necesaria la adquisición de datos para su buen funcionamiento

5.8 ELEMENTOS

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

5.9 ATRIBUTOS

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben ir entre comillas.

Por ejemplo, un elemento "nombre" puede tener un atributo "tipo" , con valor "texto".

<nombre tipo="texto">Jaimito</nombre>

5.10 ENTIDADES PREDEFINIDAS

Entidades para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado en el procesador XML.

Ejemplo: Entidad Predefinida: & Caracter &

5.11 COMENTARIOS

Comentarios a modo informativo para el programador que han de ser ignorados por el procesador.

Los comentarios en XML tienen el siguiente formato:

<!--- Esto es un comentario --->

<!-- Otro comentario -->

5.12 VALIDEZ

Que un documento esté "bien formado" solamente se refiere a su estructura sintáctica básica, es decir, que se componga de elementos, atributos y comentarios como XML especifica que se escriban. Ahora bien, cada aplicación de XML, es decir, cada lenguaje definido con esta tecnología, necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento.

Esta relación entre elementos se especifica en un documento externo o definición (expresada como DTD *(Document Type Definition = Definición de Tipo de Documento)* o como XSchema). Crear una definición equivale a crear un nuevo lenguaje de marcado, para una aplicación específica.

5.13 XML SCHEMAS (XSD)

Un Schema es algo similar a un DTD. Define qué elementos puede contener un documento XML, cómo están organizados y qué atributos y de qué tipo pueden tener sus elementos.

5.14 VENTAJAS DE LOS SCHEMAS FRENTE A LOS DTDs

- Usan sintaxis de XML, al contrario que los DTDs.
- Permiten especificar los tipos de datos.
- Son extensibles.

6. XSL

6.1 INTRODUCCIÓN

XSL (siglas de Extensible Stylesheet Language, expresión inglesa traducible como "lenguaje extensible de hojas de estilo") es una familia de lenguajes basados en el estándar XML que permite describir cómo la información contenida en un documento XML cualquiera debe ser transformada o formateada para su presentación en un medio.

Esta familia está formada por tres lenguajes:

- XSLT (siglas de Extensible Stylesheet Language Transformations, lenguaje de hojas extensibles de transformación), que permite convertir documentos XML de una sintaxis a otra (por ejemplo, de un XML a otro o a un documento HTML).

- XSL-FO (lenguaje de hojas extensibles de formateo de objetos), que permite especificar el formato visual con el cual se quiere presentar un documento XML, es usado principalmente para generar documentos PDF.

- XPath, o XML Path Language, es una sintaxis (no basada en XML) para acceder o referirse a porciones de un documento XML.

Estas tres especificaciones son recomendaciones oficiales del W3C.

Desde el 2005 ya son soportadas por algunos navegadores, como por ejemplo Mozilla o Internet Explorer, aunque, en su lugar, se pueden usar las CSS que son 100% compatibles aunque con una codificación diferente.

6.2 ¿QUÉ ES XSL?

XSLT o Transformaciones XSL es un estándar de la organización W3C que presenta una forma de transformar documentos XML en otros e incluso a formatos que no son XML. Las hojas de estilo XSLT - aunque el término de hojas de estilo no se aplica sobre la función directa del XSLT - realizan la transformación del documento utilizando una o varias reglas de plantilla. Estas reglas de plantilla unidas al documento fuente a transformar alimentan un procesador de XSLT, el que realiza las transformaciones deseadas poniendo el resultado en un archivo de

salida, o, como en el caso de una página web, las hace directamente en un dispositivo de presentación tal como el monitor del usuario.

Actualmente, XSLT es muy usado en la edición web, generando páginas HTML o XHTML. La unión de XML y XSLT permite separar contenido y presentación, aumentando así la productividad.

En el caso de nuestro proyecto esta tecnología ha sido usada para transformar los currículos de XML a Latex, como paso previo a la generación de un PDF.

7. HTML

7.1 INTRODUCCIÓN

Tim Berners-Lee fue el autor principal de HTML, asistido por sus colegas del CERN, una organización científica internacional con sede en Ginebra, Suiza.

7.2 ¿QUÉ ES HTML?

HTML es un lenguaje muy sencillo, una aplicación del SGML(Standard Generalized Markup Language) para definir tipos de documentos estructurados y lenguajes de marcas para representar esos mismos documentos.HTML es el acrónimo de HyperText Markup Language (Lenguaje de Marcadode Hipertexto) .Es el lenguaje que sirve para crear las páginas web. Permite describir hipertexto, es decir, texto presentado de forma estructurada con enlaces (hyperlinks) que conducen a otros documentos o fuentes de información relacionadas.

Por tanto es una forma de presentar objetos o textos en un visor, así que es lo mismo el ordenador que usemos o el sistema operativo, siempre que tenga un navegador que admita este formato. Hace posible presentar información (por ejemplo, investigaciones científicas) en Internet.

El lenguaje HTML contiene dos partes:

- El contenido, que es el texto que se verá en la pantalla de un ordenador,
    - Etiquetas y atributos que estructuran el texto de la página web en encabezados, párrafos, listas, enlaces, etc. y normalmente no se muestra en pantalla.
    - Las etiquetas, que son un conjunto de caracteres que rodean partes del documento, están formadas por el símbolo.

8. CSS

8.1 INTRODUCCIÓN A LAS CSS

El lenguaje HTML está limitado a la hora de aplicarle forma a un documento, ya que fue concebido para otros usos.

Para solucionar estos problemas los diseñadores han utilizado técnicas tales como la utilización de tablas para ajustarlas, utilización de etiquetas que no son estándares del HTML y otras. Esto ha causado a menudo problemas en las páginas a la hora de su visualización en distintas plataformas.

Además, los diseñadores se han encontrado con gran dificultad a la hora de maquetar las páginas, ya que en muchas ocasiones se daba forma a las páginas sobre el papel, donde el control sobre la forma del documento es absoluto

Finalmente, otro antecedente que ha hecho necesario el desarrollo de esta tecnología consiste en que las páginas web tienen mezclado en su código HTML el contenido del documento con las etiquetas necesarias para darle forma. Desde el punto de vista de la riqueza de la información y la utilidad de las páginas a la hora de almacenar su contenido, es un gran problema que estos textos estén mezclados con etiquetas incrustadas para dar forma a estos: se degrada su utilidad.

8.2 CARACTERISTICAS Y VENTAJAS DE LAS CSS

El modo de funcionamiento de las CSS define, mediante una sintaxis especial, la forma de presentación que le aplicaremos a:

- Una web, de modo que se puede definir la forma de toda la web de una sola vez.
- Un documento HTML, se puede definir la forma, en un pequeño trozo de código en la cabecera, a toda la página.
- Una parte del documento, aplicando estilos visibles en un trozo de la página.

- Una etiqueta, llegando incluso a poder definir varios estilos diferentes para una sola etiqueta.

Entre las ventajas de las CSS cabe destacar las siguientes acciones:

- Se puede definir la distancia entre líneas del documento.
- Se puede colocar elementos en la página con mayor precisión, y sin lugar a errores.
- Se puede definir, márgenes, subrayados, tachados...
- Se pueden utilizar más unidades para definir atributos en las páginas.

8.3 NAVEGADORES QUE LO SOPORTAN

Esta tecnología es bastante nueva, sólo los navegadores de Netscape versiones de la 4 en adelante y de Microsoft a partir de la versión 3 son capaces de soportar las CSS.

8.4 APLICACIONES DE LAS CSS

- Para definir estilos en pequeñas secciones se utiliza <SPAN> con su atributo style.
- Para definir estilos para un etiqueta utilizamos el atributo style.
- Para definir un estilo para una parte de una página, se utiliza la etiqueta <DIV> con su atributo style.
- Para definir estilos para que sean aplicados a toda la página:
- Se pueden definir estilos de todo un sitio web. Esto se consigue creando un archivo donde tan sólo colocamos las declaraciones de estilos de la página y enlazando todas las páginas del sitio con ese archivo, de manera que todas las páginas compartan una misma declaración de estilos.

Para la aplicación de las CSS, se debe tener en cuenta lo siguiente:

- los estilos se heredan de una etiqueta a otra, es decir, estas declaraciones también afectarán a etiquetas que estén dentro de todo el cuerpo.
- En muchas ocasiones más de una declaración de estilos afecta a la misma porción de la página. Siempre se tiene en cuenta la declaración más particular. Pero las declaraciones de estilos se pueden realizar de múltiples modos y con varias etiquetas. Entre estos modos hay una jerarquía que a continuación se muestra:
    - Declaración de estilos con fichero externo. (Para todo un sitio web)
    - Declaración de estilos para toda la página. (Con la etiqueta <STYLE> en la cabecera de la página)
    - Definidos en una etiqueta en concreto. (Utilizando el atributo style en la etiqueta en cuestión)
    - Declaración de estilo para una porción pequeña del documento. (Con la etiqueta <SPAN>)

9. JAVASCRIPT

9.1 INTRODUCCIÓN

Javascript nació con la necesidad de permitir a los autores de sitio web crear páginas que permitan interactuar con los usuarios, ya que el HTML solo permitía crear páginas estáticas donde se podía mostrar textos con estilos, pero se necesitaba interactuar con los usuarios.
En los años de 1990, Netscape creo Livescript; las primeras versiones de este lenguaje estaban vinculadas a diseñadores Web que no necesitaban utilizar un compilador.

En diciembre de 1995, Netscape y Sun Microsystems (el creador del lenguaje Java) reintroducen un lenguaje con el nombre de Javascript. En respuesta a la popularidad de Javascript, Microsoft lanzo su propio lenguaje de programación a base de script, VBScript (una pequeña versión de Visual Basic).

En el año de 1996 Microsoft se interesa por competir con Javascript por lo que lanza su lenguaje llamado Jscript, introducido en los navegadores de Internet Explorer. Desde que los navegadores incluyen el Javascript, no necesitamos el Java Runtime Environment (JRE), para que se ejecute.

9.2 ¿QUÉ ES JAVASCRIPT?

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

A pesar de su nombre, JavaScript no guarda ninguna relación directa con el lenguaje de programación Java. Legalmente, JavaScript es una marca registrada de la empresa Sun Microsystems.

9.3 NAVEGADORES QUE LO SOPORTAN

Javascript es soportado por la mayoría de los navegadores como Internet Explorer, Netscape, Mozilla Firefox, entre otros.
Con el surgimiento de lenguajes como PHP del lado del servidor y Javascript del lado del cliente, surgió Ajax en acrónimo de (Asynchronous Javascript And XML). El mismo es una técnica para crear aplicaciones web interactivas. Este lenguaje combina varias tecnologías:

- HTML y Hojas de Estilos CSS para generar estilos.
- Implementaciones ECMAScript, uno de ellos es el lenguaje Javascript.
- XMLHttpRequest es una de las funciones más importantes que incluye, que permite intercambiar datos asincrónicamente con el servidor web, puede ser mediante PHP, ASP, entre otros.

Debemos tener en cuenta que aunque Javascript sea soportado en gran cantidad de navegadores nuestros usuarios pueden elegir la opción de Activar/Desactivar el Javascript en los mismos.

9.4 SERVICIOS DE JAVASCRIPT

Entre los diferentes servicios que se encuentran realizados con Javascript en Internet se encuentran:

- Correo
- Chat
- Buscadores de Información

También podemos encontrar o crear códigos para insertarlos en las páginas como:

- Reloj
- Contadores de visitas
- Fechas
- Calculadoras
- Validadores de formularios

9.5 CARACTERíSTICAS DEL LENGUAJE

Su sintaxis es similar a la usada en Java y C. Es un lenguaje interpretado por el navegador, no se necesita tener instalado ningún Framework.

- Variables: `var = “Hola”, n=103`
- Condiciones: `if(i<10){ … }`
- Ciclos: `for(i; i<10; i++){ … }`
- Arreglos: `var miArreglo = new Array(“12”, “77”, “5”)`
- Funciones: Propias del lenguaje y predefinidas por los usuarios
- Comentarios para una sola línea: `// Comentarios`
- Comentarios para varias lineas:

```
/*
Comentarios
*/
```

- Permite la programación orientada a objetos: `docu-ment.write("Hola");`
- Las variables pueden ser definidas como: string, integer, flota, bolean simplemente utilizando “`var`”. Podemos usar “`+`” para concatenar cadenas y variables.

10.LATEX

10.1 ¿QUÉ ES LATEX?

El látex es un sistema de preparación de documentos. Preparación de documentos con látex normalmente consiste en utilizar un editor de textos (como Emacs, vi, o incluso el Bloc de notas) para modificar un látex *archivo de origen*, que tiene la extensión tex., y luego ejecutar el látex programa para convertir el archivo de origen a un documento formato de intercambio como Postscript o PDF. Una vez que el documento está en un formato de intercambio de documentos, puede ser visto de antemano en la pantalla, enviados a terceros, imprimir, etc

10.2 CARACTERISTICAS DEL LENGUAJE

- Composición artículos de revistas, informes técnicos, libros y presentaciones de diapositivas.
- El control de documentos de gran tamaño que contiene el corte, referencias cruzadas, tablas y figuras.
- Composición de las complejas fórmulas matemáticas.
- Composición de las matemáticas avanzadas con AMS-LaTeX.
- Generación automática de bibliografías e índices.
- Composición en varios idiomas.
- La inclusión de obras de arte, y de colores especiales.
- Uso de fuentes PostScript o METAFONT.

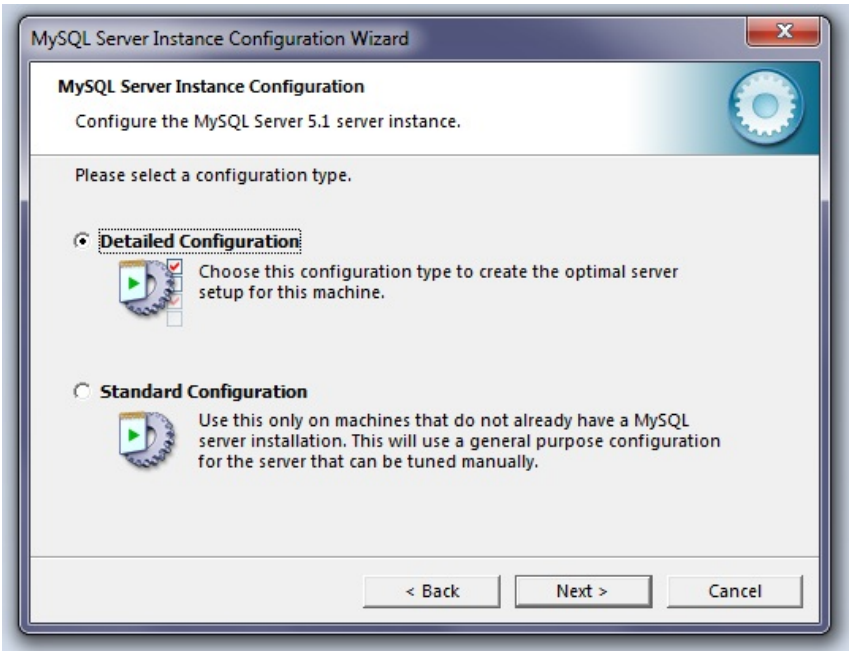
11.MANUAL DE INSTALACIÓN

11.1 MySQL

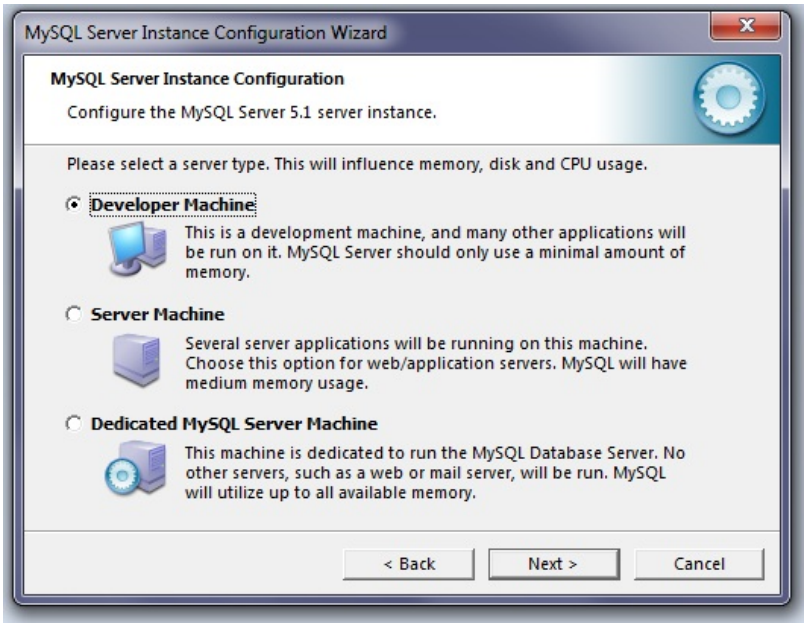
Comenzaremos por instalar el gestor de base de datos. Éste se encargará de almacenar la base de datos de los usuarios de la aplicación.

Como primer paso, descargar de la página http://dev.mysql.com/downloads/mysql/ el instalador de MySQL Community Server.

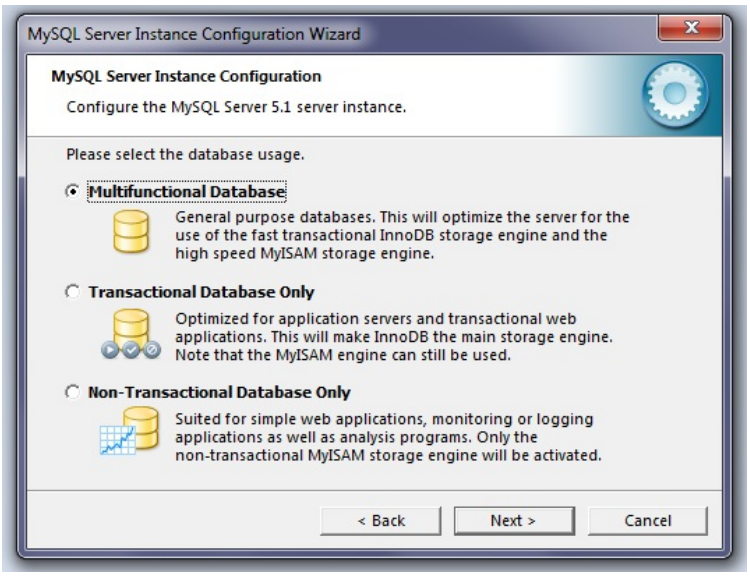
Tras ejecutar el archivo solo hay que seguir unos sencillos pasos para conseguir poner en marcha el servidor. A continuación se detallan los pasos más relevantes:



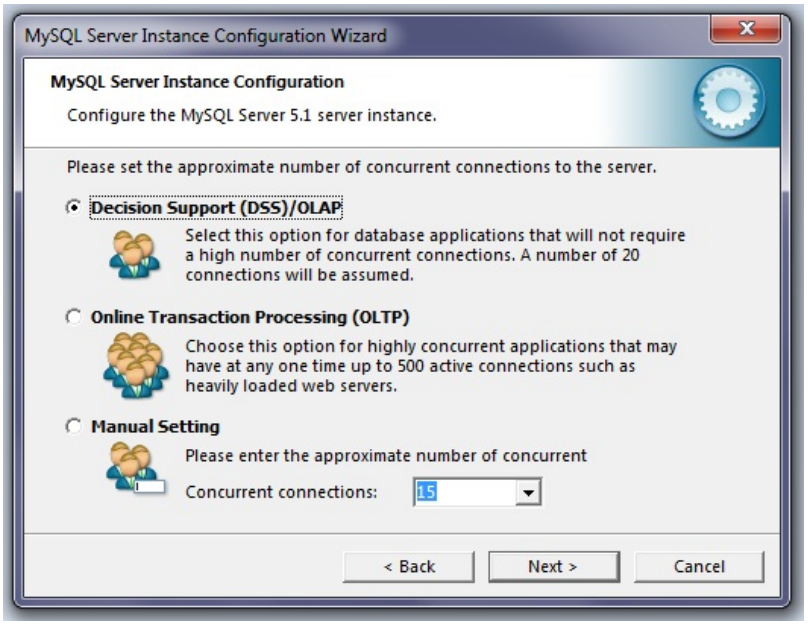
Seleccionar “Detailed Configuration” para tener un mayor control sobre el proceso de instalación y poder ajustarlo a nuestras necesidades.



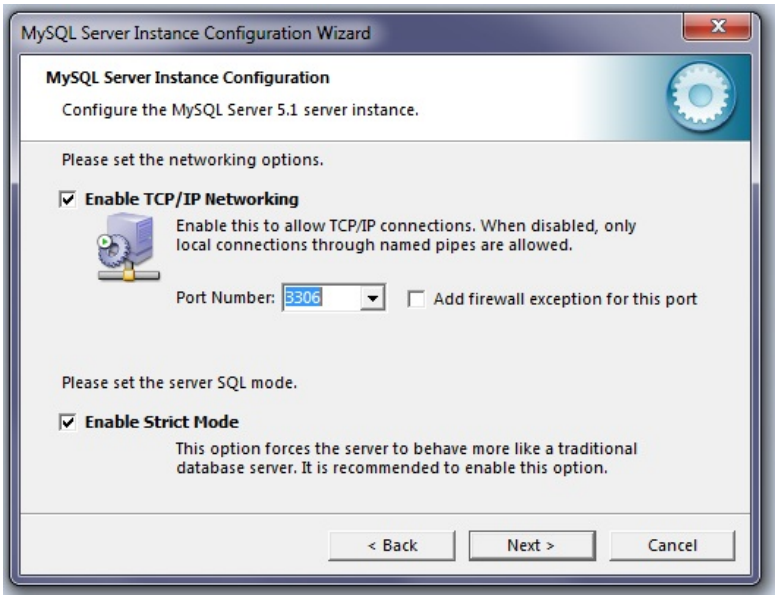
En este paso debemos decidir si la base de datos se utilizará en un equipo de desarrollo (para un uso particular de la aplicación) o en un servidor porque queremos publicar la web en internet.
Por defecto: “Developer Machine”



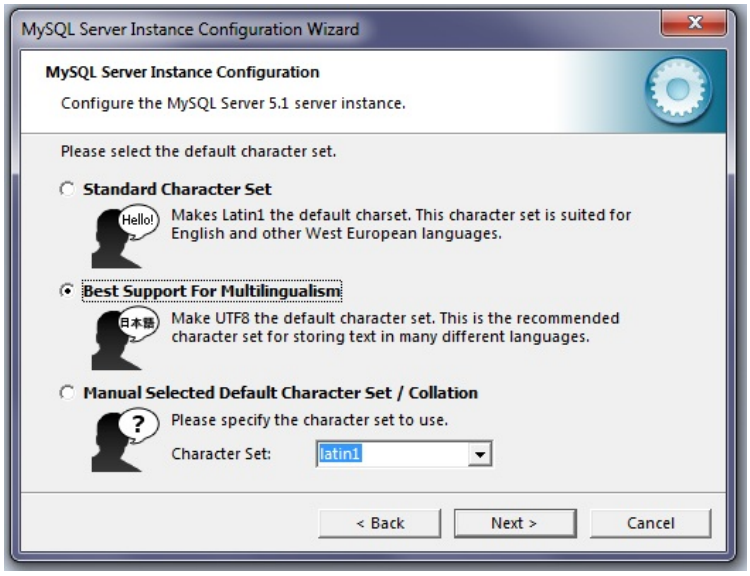
Para el uso de la aplicación solo se necesita la “Non-Transitional Database” sin embargo, si MySQL se va a utilizar también para otros usos probablemente la opción más adecuada sea la de “Multifunctional Database” que puede hacer uso de InnoDB y de MyISAM simultáneamente.



Para usar nuestra aplicación podemos seleccionar la primera opción “Decission Support” o bien seleccionar la última “Manual Setting” y establecer un valor apropiado si va a ser usada por más aplicaciones.



Este paso también depende del uso que vayamos a hacer. Para un uso particular de la aplicación no es necesario activar las conexiones TCP/IP ya que todo se realizará localmente.



Por coherencia con nuestra aplicación que se ha desarrollado haciendo uso de archivos en formato UTF-8, la mejor opción es la segunda “Best Support For Multilingualism”.

Aquí lo importante es decidir si queremos que se instale como servicio o no. Para un uso particular y esporádico de la aplicación no es tan recomendable instalarlo como servicio, podemos hacerlo por una cuestión de comodidad para no tener que preocuparnos de arrancar MySQL a mano. Sin embargo en un servidor si es recomendable, activando también la opción de que se arranque automáticamente cada vez que se inicie el equipo.

Este es un paso muy importante. Para que funcione la aplicación es necesario poner como contraseña “root”. Dependiendo del uso que le demos a MySQL puede ser interesante activar la opción de habilitar el acceso desde maquinas remotas, para un uso particular es recomendable no seleccionar esta opción.

Los posibles pasos que no se mencionan lo recomendable es dejarlos con los valores por defecto o bien su configuración no afecta para el correcto funcionamiento de la aplicación.

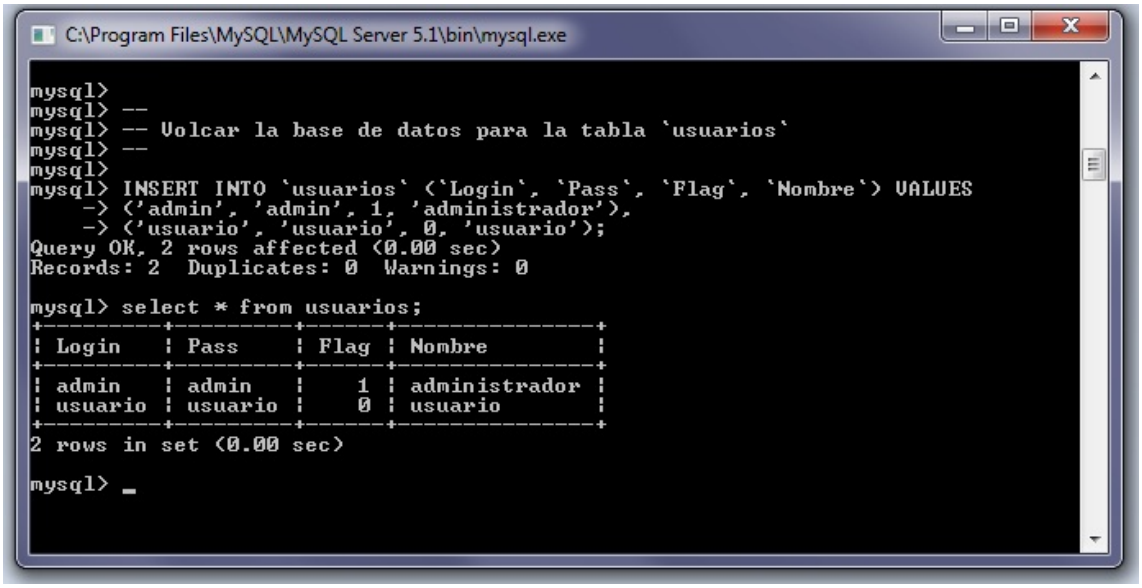
Tras esta instalación deberíamos tener ya todo listo para crear la base de datos necesaria para la aplicación.

Abrimos la consola de MySQL y ejecutamos el siguiente script:

```
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";

CREATE DATABASE `bbdd_curriculum` DEFAULT CHARACTER SET utf8 COLLATE utf8_spanish_ci;

USE `bbdd_curriculum`;

CREATE TABLE IF NOT EXISTS `usuarios` (

  `Login` varchar(16) COLLATE utf8_spanish_ci NOT NULL,

  `Pass` varchar(16) COLLATE utf8_spanish_ci NOT NULL,

  `Flag` tinyint(1) NOT NULL,
```

```
 `Nombre` varchar(20) COLLATE utf8_spanish_ci NOT NULL,
 PRIMARY KEY (`Login`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_spanish_ci;
INSERT INTO `usuarios` (`Login`, `Pass`, `Flag`, `Nombre`) VALUES
('admin', 'admin', 1, 'administrador'),
('usuario', 'usuario', 0, 'usuario');
```

Si ejecutamos una consulta simple sobre la tabla que acabamos de crear el resultado debería ser el siguiente.



También es posible hacer uso de algún gestor gráfico para importar el script pero no es en absoluto necesario.

Con esto estaría todo listo en cuanto a bases de datos.

Aunque las capturas que aquí aparecen se han realizado en un entorno Windows, el paquete de instalación en Linux es similar y tiene los mismos pasos aquí explicados.

11.2 Java

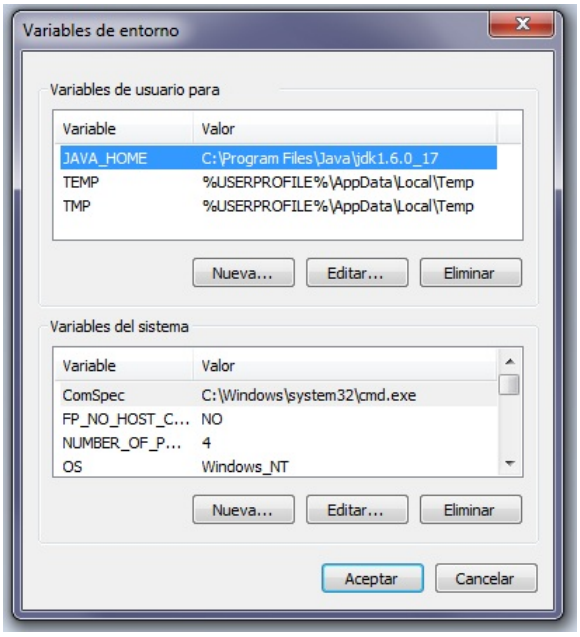
A continuación es necesario instalar el JDK o el JRE de Java. Esto instalará la máquina virtual de Java que será la encargada de ejecutar Tomcat.

Descargar de la página http://java.sun.com/javase/downloads/index.jsp el instalador de Java y ejecutalo. Sigue el asistente para completar la intalación.

Antes de instalar Tomcat es muy importante establecer una variable de entorno que le indique posteriormente a Tomcat donde se encuentra instalado Java.

Si has instalado JRE, establece una variable de entorno llamada JRE_HOME a la ruta donde instalaste Java. Por ejemplo “C:\Program Files\jre6.0” o “/usr/local/java/jre6.0”.

Si has instalado JDK, establece una variable de entorno llamada JAVA_HOME a la ruta donde lo instalaste. Por ejemplo “C:\Program Files\j2sdk6.0” o “/usr/local/java/j2sdk6.0”.



Esta debería ser la situación actual en un entorno Windows.

En linux hay varias formas de crear una variable de entorno, pero la más sencilla puede que sea editar el archivo /home/[nombre de usuario]/.bashrc (Este archivo puede variar según la distribución de Linux) y añadir al final esto.

export JAVA_HOME=ruta de java

Otra opción es añadirla en /etc/profile o en cualquier script que arranque al inicio de los de /etc/init.d.

11.3 Tomcat

A continuación hay que instalar Tomcat. Éste es el servidor elegido para procesar las páginas jsp de la aplicación.

Como primer paso, descargar de la página http://tomcat.apache.org/ la distribución binaria de Tomcat adecuada a nuestro sistema operativo.

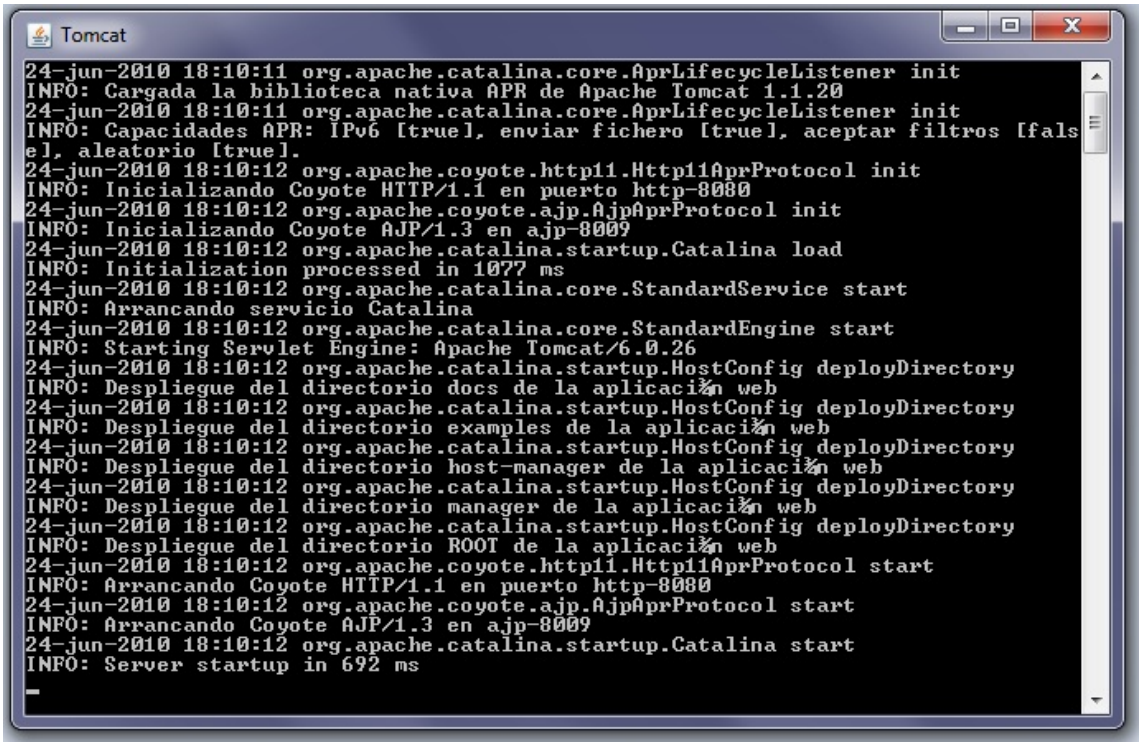
Descomprimir el archivo en una carpeta. Por ejemplo “C:/Program Files/”.

En el momento de realizar este manual la versión de Tomcat actual es la 6.0.26. En versiones anteriores era necesario definir una variable de entorno “$CATALINA_HOME” que apunta al directorio donde hemos instalado Tomcat.

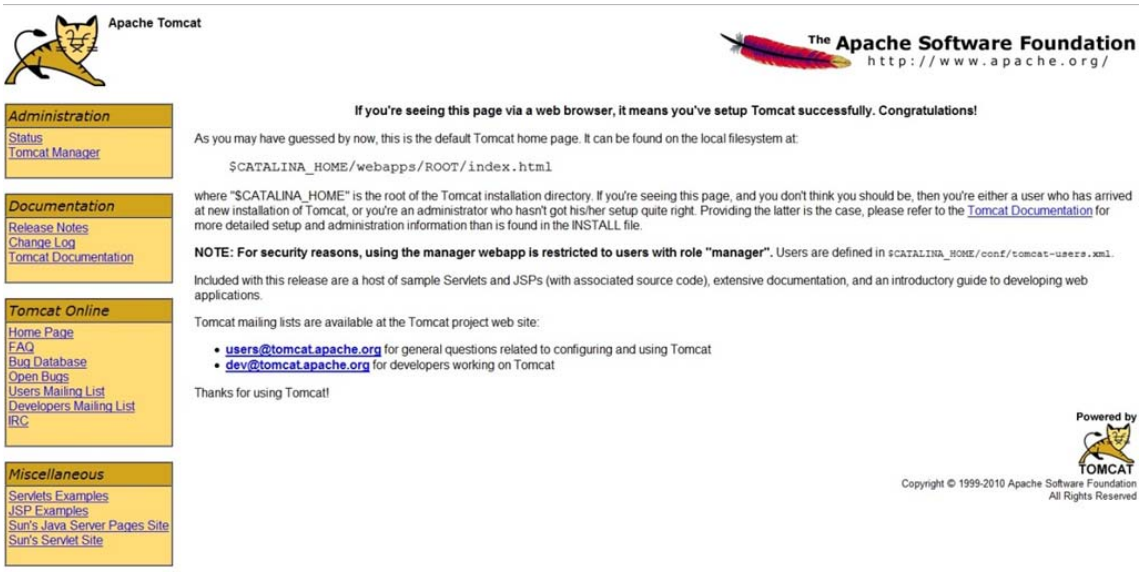
Existe la posibilidad instalarlo como un servicio, lo cual puede ser interesante en al caso de que se instale en un servidor permanente. En distribuciones Linux también es posible instalarlo a través de algún gestor de paquetes incorporado.

Para arrancar el servidor es necesario ejecutar el archivo de script situado en “$CATALINA_HOME/bin”: startup.bat (Windows) o startup.sh (Linux).

Si el servidor arranca bien el resultado tiene que ser similar a este:



Para comprobar que todo esta funcionando correctamente introducimos en el navegador web que usemos habitualmente http://localhost:8080



Deberíamos llegar a una pantalla similar a esta. Significa que todo funciona como debería.

Para apagar Tomcat ejecutaremos de “$CATALINA_HOME/bin” el script shutdown.bat (Windows) o shutdown.sh (Linux).

Como parte opcional y recomendada de la instalación, podemos crear un usuario que nos permita acceder al Tomcat Manager, con el que podemos gestionar nuestras páginas.

El procedimiento consiste en rellenar el archivo “tomcat-users.xml” situado en “$CATALINA_HOME/conf” dejándolo con el siguiente contenido:

```
<tomcat-users>
    <role rolename="manager"/>
    <user username="tomcat" password="tomcat" roles="manager"/>
</tomcat-users>
```

Con esto finalizamos la instalación de Tomcat y ya podemos desplegar la aplicación en el servidor.

11.4 Despliegue de la web en Tomcat

Ahora que ya está todo listo solo falta desplegar la página web en Tomcat.

La aplicación web completa está empaquetada en un archivo único llamado Curriculum.war

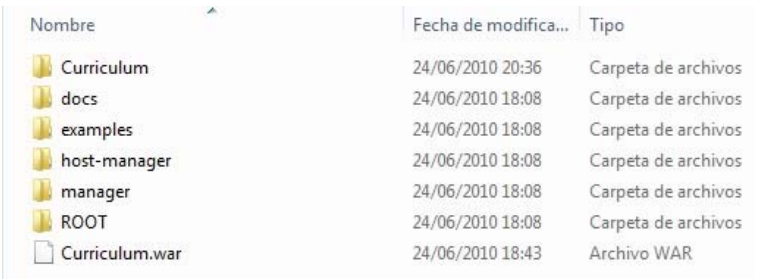
La estructura de un Archivo WAR es la siguiente:

- **/WEB-INF/web.xml** : Contiene elementos de seguridad de la aplicación así como detalles sobre los Servlets que serán utilizados dentro de la misma.
- **/WEB-INF/classes/** : Contiene las clases Java adicionales a las del JDK que serán empleadas en los JSP's y Servlets
- **/WEB-INF/lib/** : Contiene las librerías, los JAR's que serán utilizados por la aplicación.
- **/META-INF/ :** Directorio relacionado con los ficheros .jar. Contiene el manifiesto (la lista de contenidos) de un jar. Las aplicaciones web empaquetadas en un .war, son casos especiales de .jars. En teoria se puede borrar de forma segura.
- **/ *.html *.jsp *.css y otros directorios**: El directorio raíz y los demás directorios contienen los elementos que comúnmente son utilizados en un sitio, Documentos en HTML , JSP's , CSS("Cascading Style Sheets") y otros elementos.

Este tipo de estructura permite portabilidad a las diversas aplicaciones que son desarrolladas en Java.

Para llevar a cabo el despliegue de la aplicación, solo es necesario copiar el archivo Curriculum.war dentro del directorio $CATALINA_HOME/webapps.

La siguiente vez que arranquemos Tomcat, la aplicación será desplegada.



Podremos acceder a la misma a través de http://localhost:8080/Curriculum

11.5 MIKTEX

Existen varias distribuciones disponibles de Latex. La que se ha utilizado para la aplicación es MIKTEX. se puede descargar de http://miktex.org

Su instalación es muy sencilla, solo es necesario seguir el asistente que proporciona.

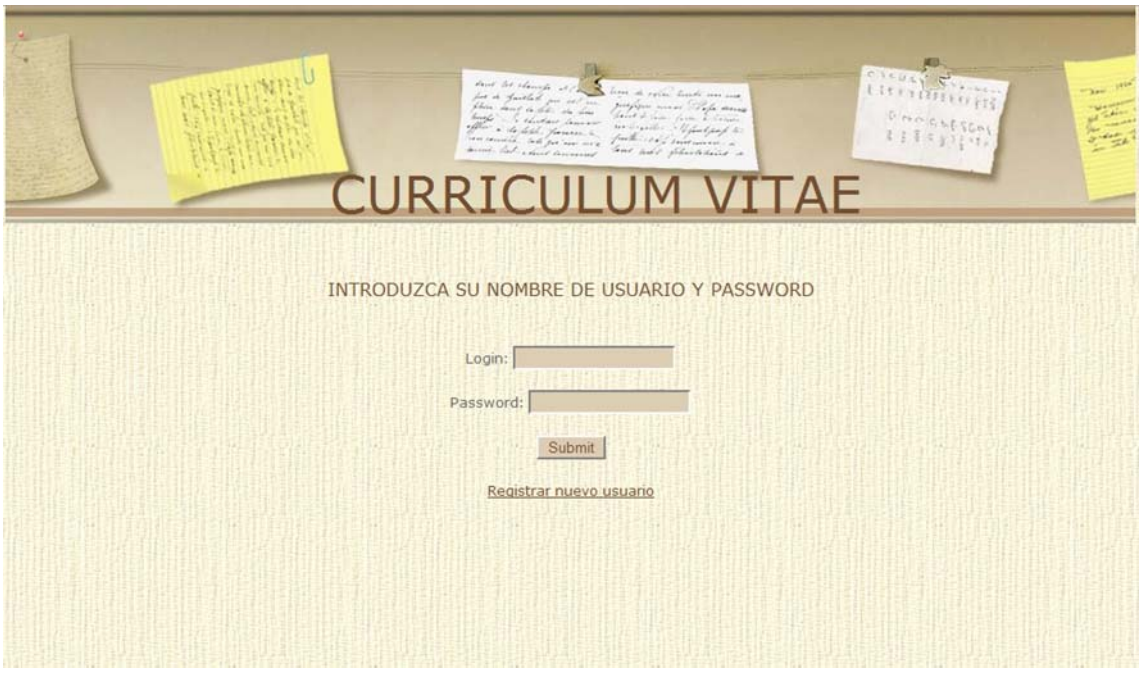
La instalación de esta parte es fundamental para la correcta generación de los currículos ya que se hace uso de varios compiladores y herramientas que incluye este paquete.

12.MANUAL DE USUARIO

Este apartado describe la funcionalidad y manejo de la web. Hay dos tipos de acceso a la misma:

1. Modo usuario.

2. Modo administrador, con más privilegios y posibilidad de creación de estilos y categorías nuevas para el currículum.



Veremos detenidamente cada una de las posibilidades a continuación.

12.1 REGISTRO DE NUEVO USUARIO

Nada más entrar en la web se nos dará la posibilidad de acceder con nuestro usuario y contraseña, en el caso de estar registrados, o de registrarnos si no disponemos de una cuenta.

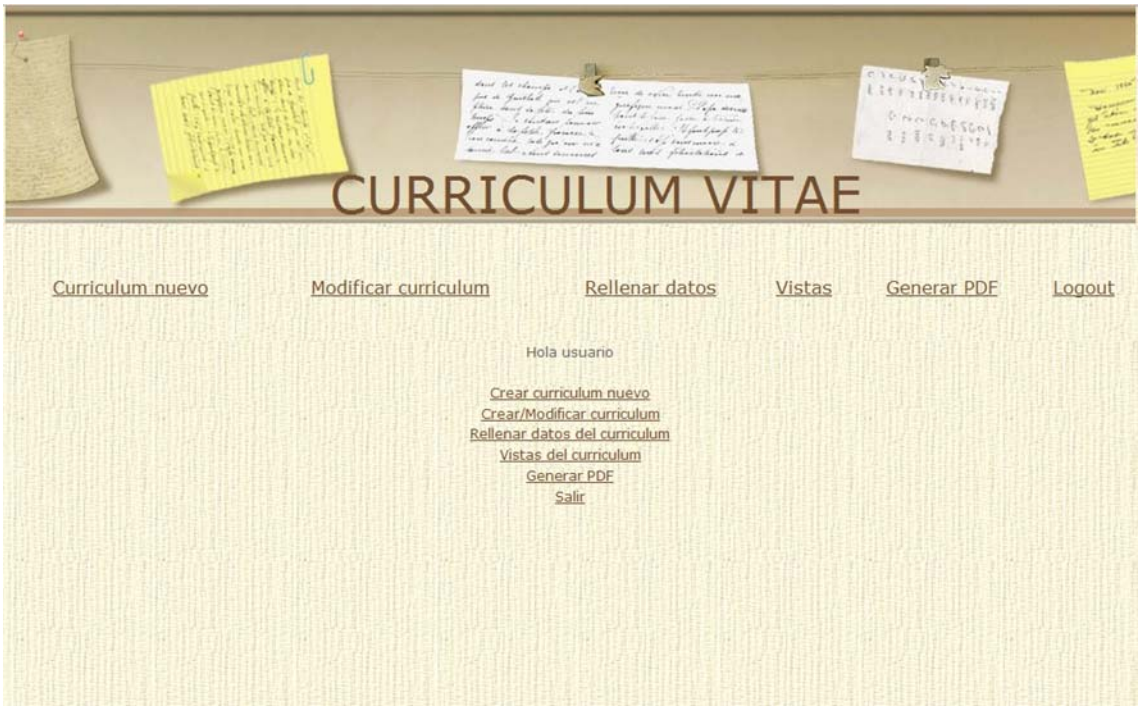
Supongamos inicialmente que no estamos registrados. Hacemos clic en "Registrar nuevo usuario" y rellenamos los apartados que aparecen.
A continuación presionamos "Registrar".
Si los datos introducidos no son incoherentes y el login indicado no está ocupado el registro se efectuará correctamente.

Ahora ya podemos acceder.

12.2 INICIO EN MODO USUARIO

Introducimos nuestro usuario y contraseña en login y password respectivamente para efectuar el acceso, seguidamente pulsamos el botón “submit”. Una vez registrados nos aparecerán las siguientes posibilidades:

- Crear currículum nuevo
- Crear/Modificar currículum
- Rellenar datos del currículum
- Vistas del currículum
- Generar PDF
- Salir

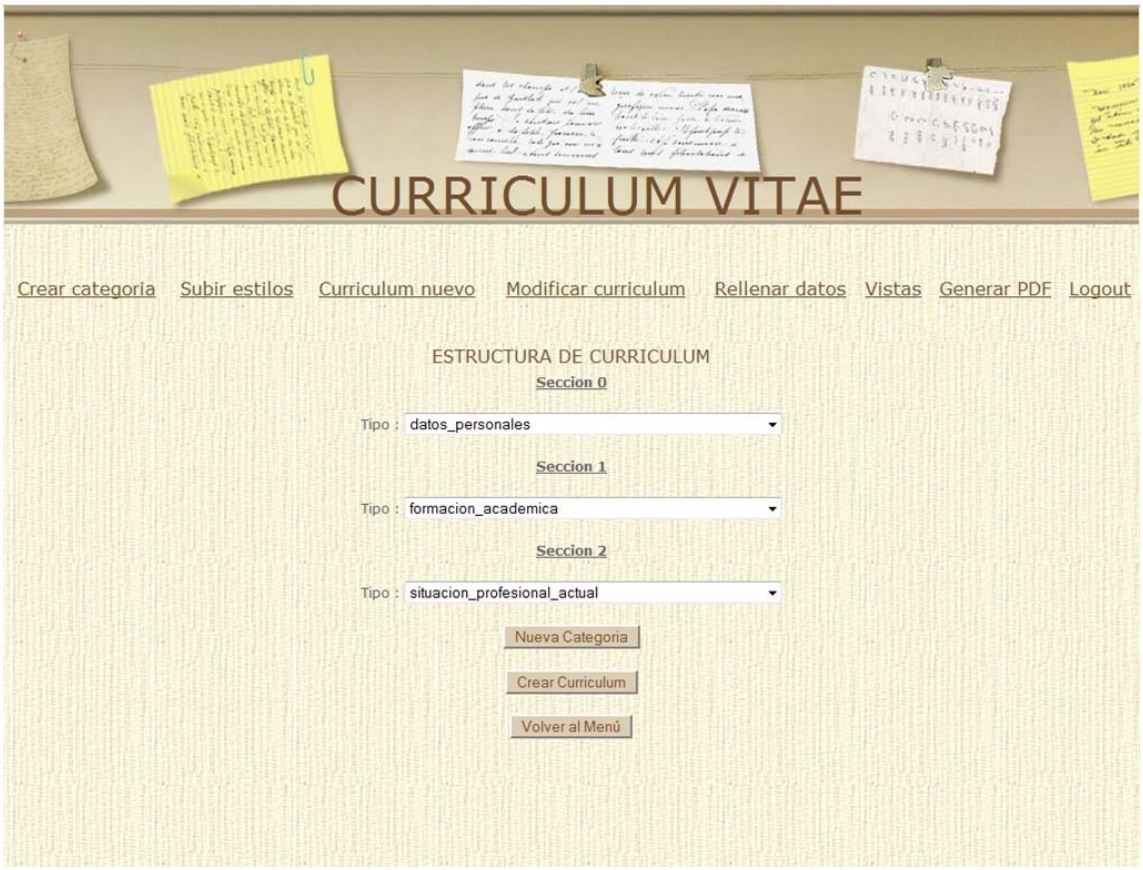


Estas opciones junto con “logout” permanecerán en la parte superior de la página como acceso directo durante toda la sesión para más comodidad. Sin embargo,

si se desea volver al menú principal, dispondremos adicionalmente de un botón “Volver al menú principal” que nos retornará a este punto.

12.2.1 Crear currículum nuevo

En este apartado se definirá la estructura del currículum. Estará compuesto por diferentes secciones o apartados, que contendrán cada una de nuestras categorías, como pueden ser los datos personales del usuario o su formación académica.



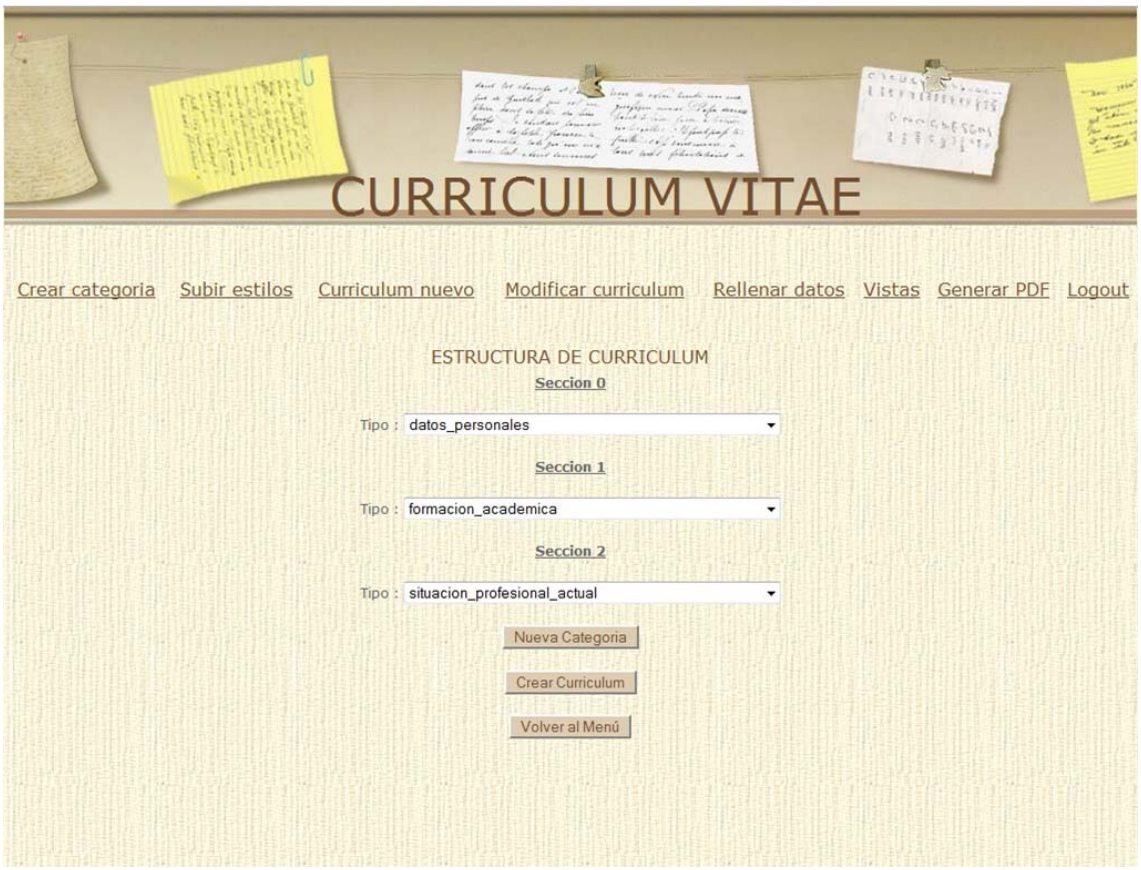
Por cada sección se selecciona de la lista desplegable el tipo deseado para la misma. Si se quiere añadir una nueva sección presione “nueva categoría” y seleccione el tipo para esta nueva sección de su currículum vitae. Este proceso se repite tantas veces como secciones diferentes desee que conformen su documento. Al finalizar pulse “Crear currículum”. Si todo esta correcto le saldrá el siguiente mensaje:

“Currículum definido correctamente”

Ahora pinche en “Volver al menú principal” para continuar editando el documento. Si pulsamos este botón antes de pulsar “Crear currículum” no se guardará la estructura creada y se asume que el usuario quiere dejar de definir la estructura.

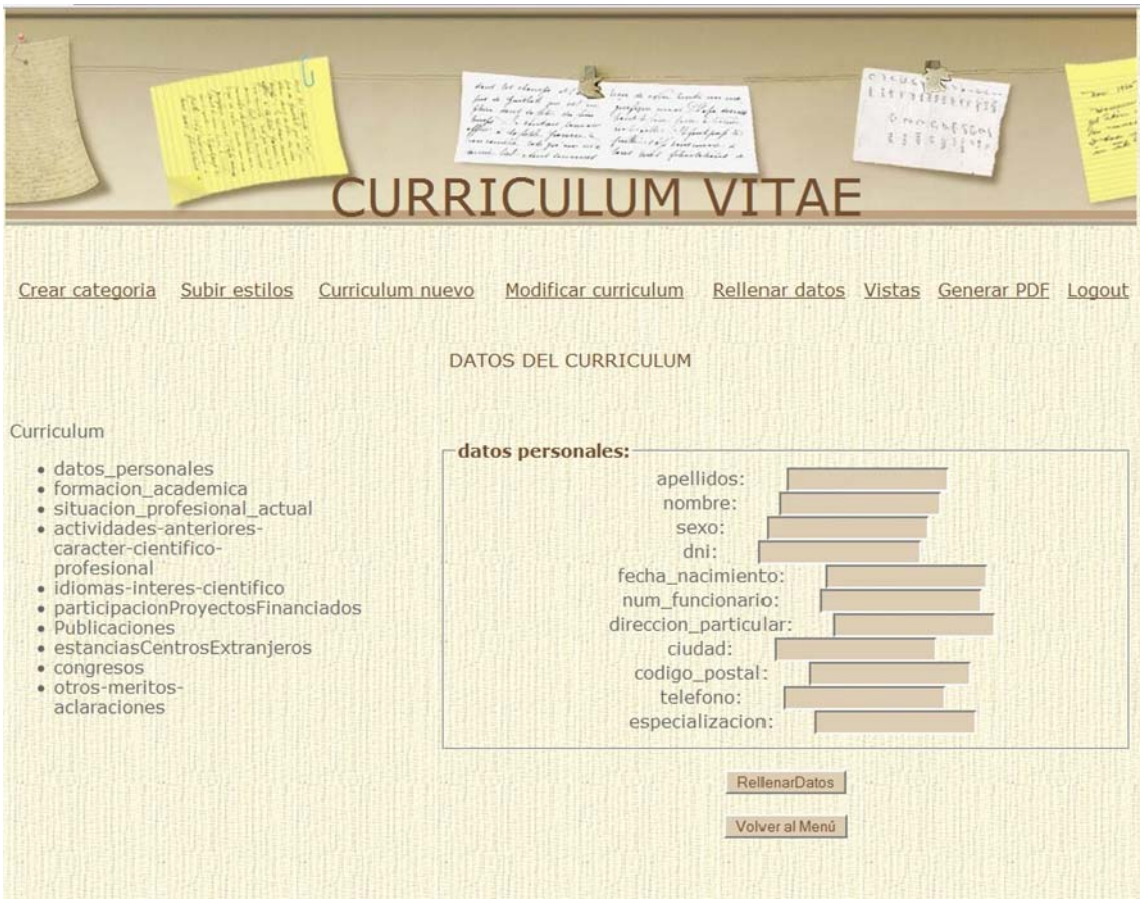
12.2.2 Crear/modificar currículum

Esta opción nos va a permitir también definir la estructura de nuestro currículum, pero partiendo de la última estructura que realizamos y que tenemos guardada de forma apropiada en nuestro usuario. Podemos cambiarla o ampliarla con alguna sección nueva.
Finalizado el proceso de modificación pinchamos en “Volver al menú principal” y después en “Volver al menú principal” como hacíamos en el apartado anterior.



12.2.3 Rellenar datos del currículum

Una vez definida la estructura solo necesitamos rellenar los datos que compondrán las secciones. Rellenamos cada apartado con nuestra información personal y luego pulsamos “Rellenar datos” que los guardará. A continuación pulsamos “Volver al menú principal” para continuar nuestro proceso.



12.2.4 Vistas del currículum

Con la estructura creada y los datos guardados podemos proceder a aplicarle estilo a cada categoría.
Para empezar le daremos un nombre a nuestra vista. A continuación vamos seleccionando el estilo deseado del desplegable del que dispone cada categoría. Esto hará que tenga un aspecto visual acorde a un determinado formato como puede ser el Europeo (estilo utilizado en los currículum de los docentes de la Universidad Complutense).
Hacemos este paso tantas veces como categorías comprenda nuestra vista. En cada desplegable tenemos la opción “no incluir”. Si elegimos dicha opción, esa categoría no será incluida en la vista.
Cuando terminamos de elegir estilos pulsamos el botón “Generar vista”. Como siempre, y si todo está correcto, nos aparecerá un mensaje indicativo de que la vista se ha generado correctamente.
Pinchamos en “Volver al menú principal”. Ya solo nos queda visualizar el documento editado.

12.2.5 Generar PDF

Con la vista de nuestro currículum ya creada podemos generar un PDF y visualizarlo pinchando en esta opción. Nada más pulsarlo nos aparecerá una ventana con la vista del documento.

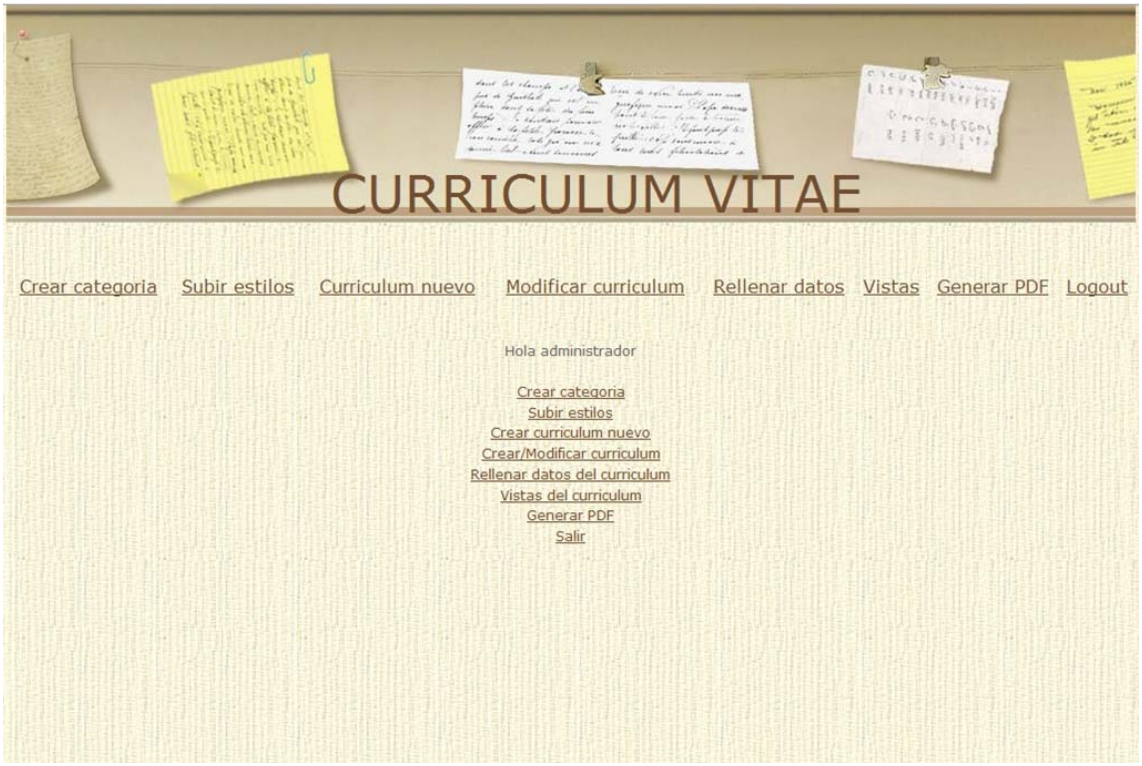
12.2.6 Salir

Ver apartado 12.4 del Manual de Usuario.

12.3 MODO ADMINISTRADOR

Introducimos nuestro usuario y contraseña de administrador en login y password respectivamente para efectuar el acceso, seguidamente pulsamos el botón “submit”. Una vez registrados nos aparecerán las siguientes posibilidades:

- Crear categorías
- Subir estilos
- Crear currículum nuevo
- Crear/Modificar currículum
- Rellenar datos del currículum
- Vistas del currículum
- Generar PDF



Como se puede observar las opciones que nos aparecen son las mismas salvo las dos primeras, crear categoría y subir estilos, que solo pueden ser accedidas por el

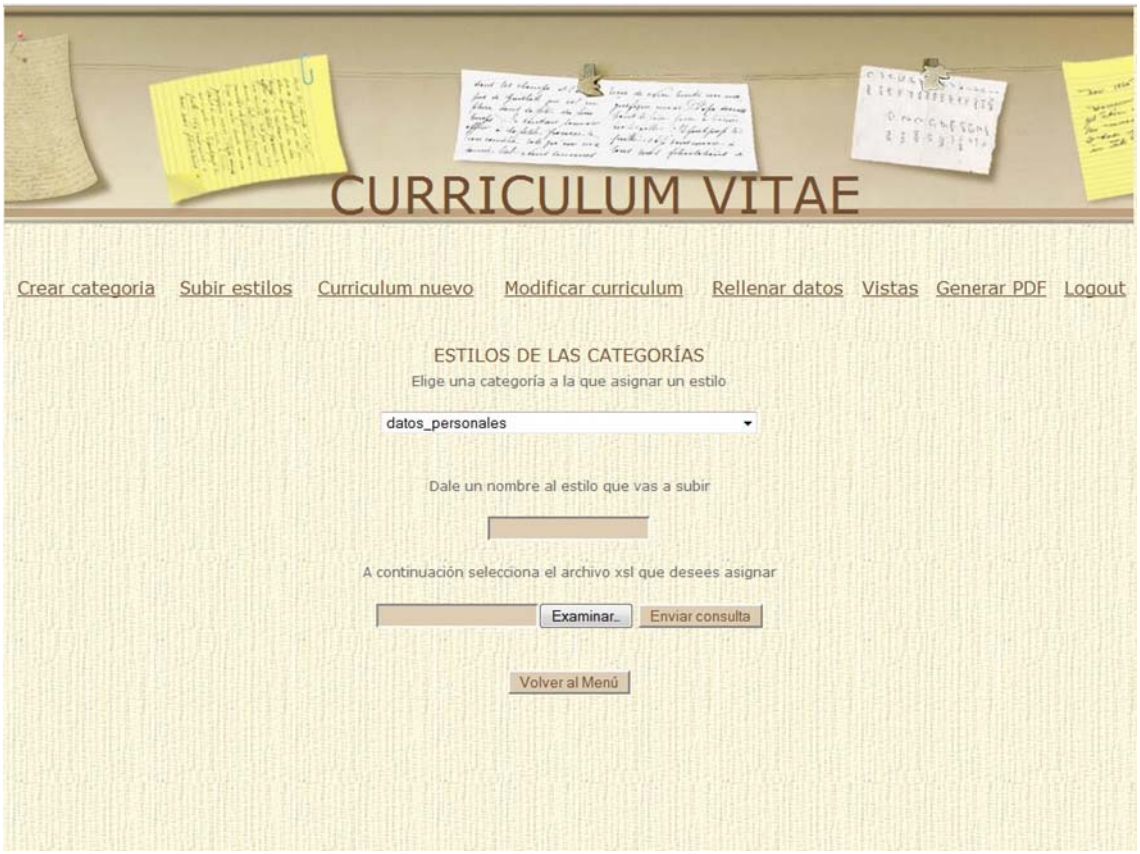
administrador de la web. A continuación veremos su funcionalidad.

12.3.1 Crear categorías

A esta opción solo podremos acceder con privilegios de administrador. Nos va a permitir definir nuevas categorías con diferentes secciones o apartados del tipo que le indiquemos en el respectivo desplegable (texto, número, fecha, otra categoría ya definida, etc.).

Para empezar daremos un nombre a nuestra categoría y, a continuación, definiremos las diferentes secciones que la compondrán. Cada sección tendrá un nombre y un tipo que la define.
Se añadirán tantas secciones diferentes como se quiera. Cuando acabemos con una y queramos añadir otra pulsamos “Nueva sección”. Cuando la categoría esté completa y no queramos añadir nuevas secciones pulsamos “Crear categoría”. Si todo está correcto nos aparecerá un mensaje: “Categoría creada correctamente”.

Al acabar puede pinchar en “Volver al menú principal”, o bien cualquiera de las opciones que permanecen en la parte superior de la página durante toda la sesión. Una vez creada todos los usuarios tendrán acceso a esta y la podrán utilizar al editar su currículum.



12.3.2 Subir estilos

Cada categoría puede tener asociados diferentes estilos. En este apartado se pueden añadir estilos a la categoría que se desee.

Solo debe elegir del desplegable una de las categorías existentes, darle un nombre a ese estilo y subir el archivo con extensión .XSL usando el botón “Examinar”.
Una vez cargado el archivo pulse “Enviar consulta”, le aparecerá el siguiente mensaje en caso de que el proceso se realice con éxito:

“Estilo subido correctamente”

12.4 FINALIZAR LA SESIÓN

Para cerrar la sesión correctamente debemos pulsar la última opción en la parte superior derecha de la página “Logout” o, si se prefiere, desde el menú principal en la opción “Salir”.

BIBLIOGRAFÍA

Leslie Lamport: LaTeX -un sistema de preparación de- la guía del usuario. Segunda edición. Addison-Wesley 1994, ISBN 0-201-52983-1.

Thinking In Java. Bruce Eckel. Prentice-Hall International Edition - 2005

http://www.w3schools.com/

http://miktex.org/

http://java.sun.com/

http://tomcat.apache.org/

http://www.jdom.org/

http://xml.apache.org/xalan-j/

http://www.mysql.com/

https://developer.mozilla.org/index.php?title=es/Guía_JavaScript_1.5/Concepto_de_JavaScript

http://www.lsi.us.es

http://antivirus.interbusca.com/glosario/ASP.html

http://scottmcpeak.com/latex/whatislatex.html

http://www.webopedia.com/TERM/L/LaTeX.html

http://es.wikipedia.org/wiki/Servidor_web

http://www.idg.es/iWorld/articulo.asp?id=119691

http://www.desarrolloweb.com/html/

Nosotros, los autores del proyecto del proyecto Generador de Currículos en diferentes estilos, de la asignatura de *Sistemas Informáticos*:

Joaquín Castilla Carramiñana con DNI: 51998447 R

Laura Mendiola Martínez con DNI: 50898405-A

Rosa Olivia Zumaeta Sánchez con DNI: X1661769-L

Dirigidos por:

Dr. Samir Genaim

Departamento de Sistemas Informáticos y Computación

Autorizamos a la Universidad Complutense de Madrid a utilizar y difundir, con fines académicos, el contenido de este documento de texto , así como el contenido del CD complementario que adjuntamos con el mismo.

Rosa Olivia Zumaeta Laura Mendiola Martínez

Joaquín Castilla Carramiñana